Όραση υπολογιστών

Φθινόπωρο 2020 – Διαμάντης Πατσίδης 57592

Εργασία 1η

Εισαγωγή

Στην παρούσα εργασία υλοποιείται ένα γραμμικό (Gaussian filter) και ένα μη-γραμμικό (Median filter) φίλτρο, τα οποία στην συνέχεια εφαρμόζονται για την αποθορυβοποίηση εικόνων . Τα αποτελέσματα εξετάζονται και έπειτα οι εικόνες υπόκεινται σε επεξεργασία, σύμφωνα με την μεθοδολογία που αναπτύχθηκε στην εργασία. Με την επεξεργασία επιτυγχάνεται η ανίχνευση όλων των υποπεριοχών κειμένου της εικόνας (π.χ. αρίθμηση σελίδας, υποσέλιδο, παράγραφος, τίτλος κ.α.), σχεδιάζοντας το περιβάλλον κουτί (bounding box) καθώς και ένα μοναδικό αύξων αριθμό της υποπεριοχής. Επίσης για την κάθε υποπεριοχή υπολογίζεται η επιφάνεια που καταλαμβάνεται από κείμενο, η επιφάνεια του περιβάλλοντος κουτιού, ο αριθμός των λέξεων καθώς και η μέση τιμή διαβάθμισης του γκρι των εικονοστοιχείων, με τέτοιο τρόπο ώστε η ταχύτητα εκτέλεσης υπολογισμού να είναι ανεξάρτητη του μεγέθους της υποπεριοχής.

Αποθορυβοποίηση

Για την αποθορυβοποίηση υλοποιήθηκε το φίλτρο Gauss καθώς και το φίλτρο Median.

Το kernel του φίλτρου Gauss στον διακριτό χώρο περιγράφεται από την παρακάτω σχέση:

$$g[m,n;\sigma] = \exp -\frac{m^2+n^2}{2\sigma^2}$$

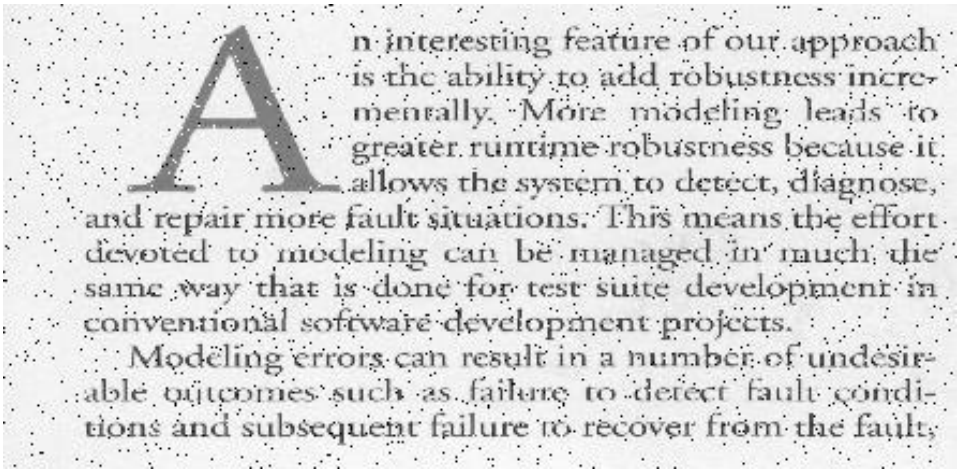
όπου m , n προσδιορίζουν την θέση του kernel και σ η τυπική απόκλιση.

Για την υλοποίηση του φίλτρου δημιουργήθηκε η συνάρτηση gaussian_filter:

```
def gaussian_filter(input, output, l = 5, sig = 1):
    center = (l - 1) // 2
    ax = np.linspace(-center, center, l)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-0.5 * (np.square(xx) + np.square(yy)) / np.square(sig))
    kernel = kernel / np.sum(kernel)

    for x in range(center, input.shape[0] - center):
        for y in range(center, input.shape[1] - center):
            output[x, y] = np.sum(kernel * input[x - center:x + center + 1, y - center:y + center + 1])

    return output
```

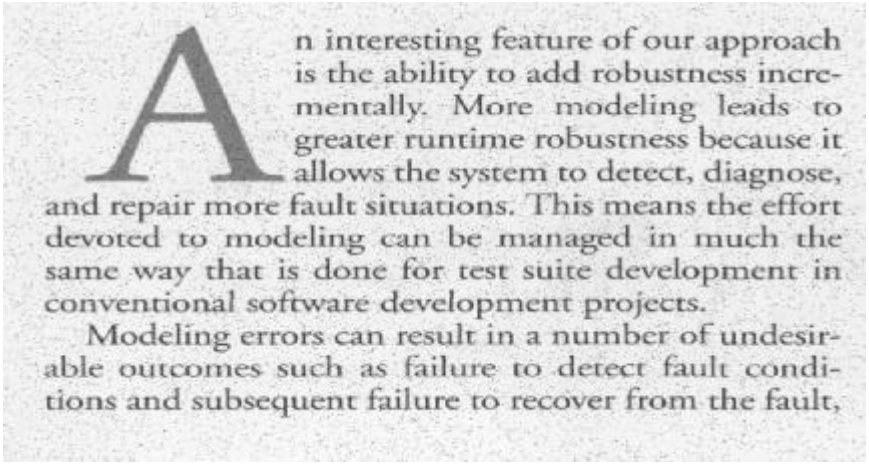
Η συνάρτηση δέχεται ως ορίσματα τέσσερις μεταβλητές, την l (αρχικοποιείται στην τιμή 5) ,την sig (αρχικοποιείται στην τιμή 1) και τις μεταβλητές input, output οι οποίες αντιπροσωπεύουν τα σήματα της εικόνας εισόδου και εξόδου αντίστοιχα του φίλτρου Gauss. Η μεταβλητή l καθορίζει το μέγεθος του kernel (συνήθης μέγεθος 5x5) και το sig την τυπική απόκλιση. Αρχικά δημιουργούνται όλοι οι δυνατοί συνδυασμοί των συντεταγμένων του kernel, m και n, οι οποίοι με βάση την παραπάνω σχέση του φίλτρου Gauss σχηματίζουν τις τιμές του ζητούμενου kernel. Στην συνέχεια, το kernel διαιρείται με το άθροισμα των στοιχείων του, ώστε όταν εφαρμοστεί στην εικόνα, η τιμή των εικονοστοιχείων να παραμείνει εντός των επιτρεπτών ορίων. Τέλος με μία επαναληπτική δομή, το φίλτρο εφαρμόζεται στο σήμα της εικόνας εισόδου και το αποθυρυβοποιημένο αποτέλεσμα προκύπτει από την διασυσχέτιση (cross-correlation) του kernel με την εικόνα εισόδου.

Παρακάτω φαίνονται τα αποτελέσματα του φίλτρου, από την εφαρμογή του σε εικόνα:


An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault;

Εικόνα 1: αρχείο «2_noise.png», πριν την αποθορυβοποίση.

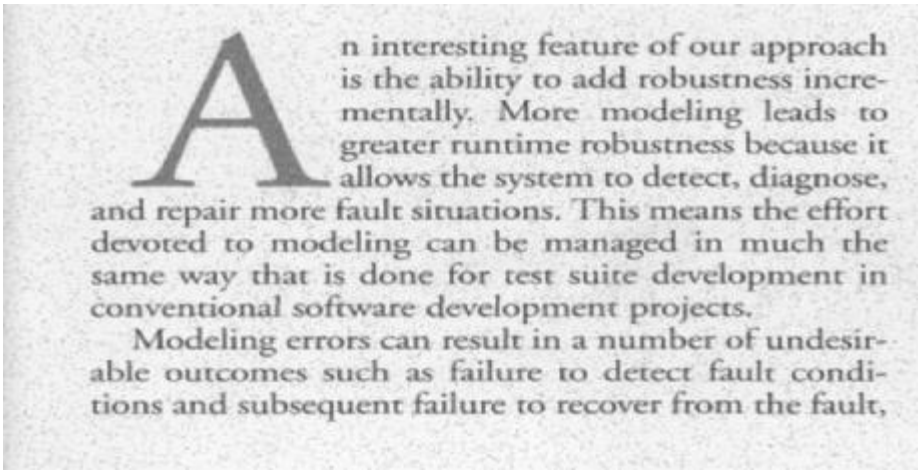

An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault,

Εικόνα 2: αρχείο «2_noise.png», μετά την αποθορυβοποίση με kernel μεγέθους 5x5 και σ=1.

Ο λόγος σήματος προς θόρυβο προκύπτει:

SNR = (original_signal / filtered_ signal) = 1.013897


An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault,

Εικόνα 3: αρχείο «2_noise.png», μετά την αποθορυβοποίση με kernel μεγέθους 5x5 και σ=10.

Ο λόγος σήματος προς θόρυβο προκύπτει:

SNR = (original_signal / filtered_ signal) = 1.013886

Για σ=10 υπήρχε μία μικρή βελτίωση στον λόγο σήματος προς θόρυβο καθώς αυτός πλησίασε στο 1:1. Ωστόσο αυτό συνοδεύτηκε με πιο έντονα θόλωμα της εικόνας και συνεπώς απώλεια στην ευκρίνεια της.

Ένα πρόβλημα που προκύπτει από την παρούσα μεθοδολογία αφορά τα εικονοστοιχεία στα περιθώρια της εικόνας, στα οποία δεν εφαρμόζεται το φίλτρο. Λόγω του τύπου των δειγμάτων (εικόνες κειμένου) και του μεγέθους τους , αυτό δεν γίνεται ιδιαίτερα αισθητό . Ωστόσο σε μικρότερου μεγέθους εικόνες οι οποίες περιέχουν διαφορετικές εντάσεις του γκρι στο περιθώριο τους μπορεί εύκολα να παρατηρηθεί. Έγινε προσπάθεια αντιμετώπισης του προβλήματος αυτού με την προσθήκη γεμίσματος τιμών του μηδενός στην εικόνα (padding) αλλά το αποτέλεσμα ήταν χειρότερο. Η μεθοδολογία του φίλτρου Gauss προσθέτει τις τιμές του αποτελέσματος διασυσχέτισης (cross correlation) του kernel με την εικόνα, συνεπώς στα όρια της εικόνας, η τιμή της πρόσθεση έτεινε στο μηδέν (χρώμα μαύρο) εξαιτίας του γεμίσματος. Μία πιθανή λύση με μεγαλύτερη επιτυχία, ίσως αποτελεί η προσθήκη γεμίσματος (padding), όχι με τιμές μηδέν αλλά με τιμές οι οποίες βασίζονται στην τοπικότητα των τιμών της περιοχής.

Το αποτέλεσμα του φίλτρου Gauss, δεν ήταν ιδιαίτερα ικανοποιητικό, έτσι για την εύρεση καλύτερου αποτελέσματος αποθορυβοποίησης, υλοποιήθηκε στην συνέχεια το φίλτρο Median. Το kernel του φίλτρου Median διατρέχει τις τιμές του σήματος της εικόνας και το κεντρικό εικονοστοιχείο αποκτάει την τιμή του εικονοστοιχείου του kernel με την μέσαία τιμή.

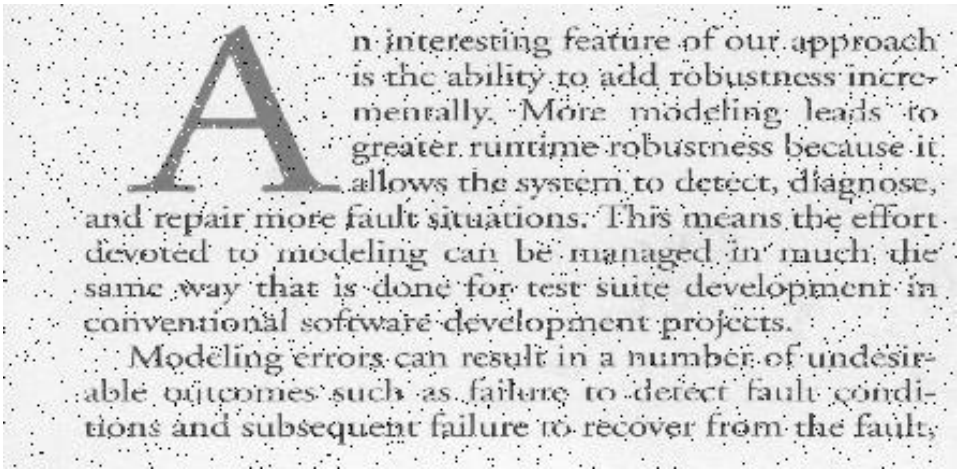
Η συνάρτηση που υλοποιεί το φίλτρο είναι η median_filter:

```
def median_filter(input, output, l = 5):
    center = (l - 1) // 2
    img = np.zeros((input.shape[0] + 2*center, input.shape[1] + 2*center))
    img[center:input.shape[0] + center, center:input.shape[1] + center] = input
    for x in range(center, img.shape[0] - center):
        for y in range(center, img.shape[1] - center):
            kernel_median = np.copy(img[x-center:x+center+1,y-center:y+center+1])
            kernel_median = kernel_median.flatten()
            kernel_median = sorted(kernel_median)
            median = kernel_median[(len(kernel_median)-1)//2]
            output[x - center,y - center] = median

    return output
```

Η συνάρτηση δέχεται τρία ορίσματα . Τα input και output που αντιπροσωπεύουν τα σήματα εισόδου και εξόδου του φίλτρου και το όρισμα l, το οποίο καθορίζει το μέγεθος του kernel. Στο σήμα εικόνας της εισόδου του φίλτρου προστίθεται γέμισμα μηδενικών (padding) ώστε να

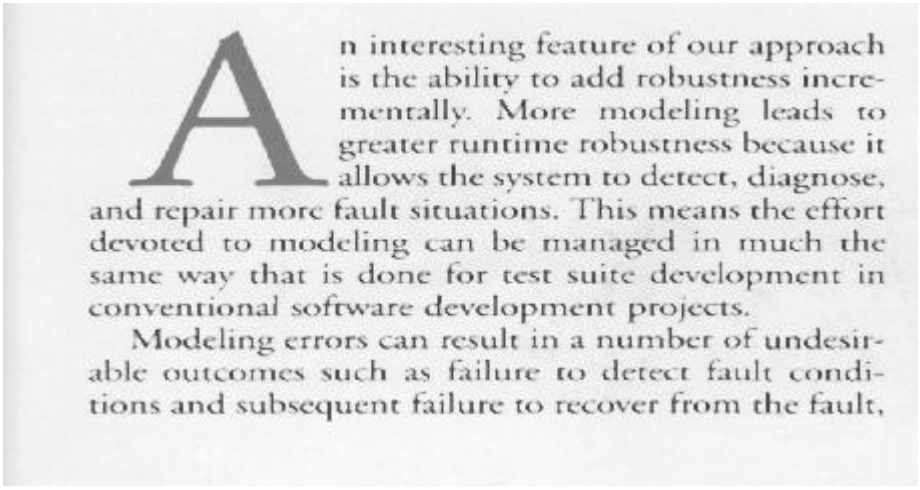
γίνει φιλτάρισμα και των ακραίων εικονοστοιχείων . Σε αντίθεση με το φίλτρο Gauss, στο φίλτρο Median η μέθοδος αυτή λειτουργεί με επιτυχία καθώς επιλέγεται η μεσαία τιμή του kernel, με αποτέλεσμα τα μηδενικά να μην επηρεάζουν το τελικό αποτέλεσμα. Αφού έχει γίνει η προσθήκη του γεμίσματος, το kernel μεγέθους l διατρέχει την εικόνα, αποκτώντας τις τιμές της περιοχής στην οποία βρίσκεται και θέτει στο κεντρικό εικονοστοιχείο τιμή ίση με την μεσαία τιμή του kernel.

Παρακάτω φαίνονται τα αποτελέσματα του φίλτρου, από την εφαρμογή του σε εικόνα:


An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault,

Εικόνα 4: αρχείο «2_noise.png», πριν την αποθορυβοποίηση.


An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault,

Εικόνα 5: αρχείο «2_noise.png», μετά την αποθορυβοποίηση με kernel μεγέθους 5x5.

Ο λόγος σήματος προς θόρυβο προκύπτει:

SNR = (original_signal / filtered_ signal) = 0.98645

Αν και ο λόγος SNR απέχει περισσότερο από τον λόγο 1:1, το αποτέλεσμα είναι αισθητά ανώτερο της μεθόδου αποθορυβοποίησης με φίλτρο Gauss. Αυτό βέβαια ήταν το αναμενόμενο, καθώς πρόκειται για θόρυβο salt and pepper για τον οποίο το φίλτρο Median παράγει καλύτερα αποτελέσματα.

Συνεπώς για την υπόλοιπη μεθοδολογία χρησιμοποιείται το φίλτρο Median. Η υλοποίηση του φίλτρου Gauss έχει συμπεριληφθεί στον τελικό κώδικα μόνο για τίς ανάγκες της εξέτασης και ελέγχου του κώδικα.

Εντοπισμός υποπεριοχών

Μετά την αποθορυβοποίηση της εικόνας για τον εντοπισμό των υποπεριοχών κειμένου υλοποιήθηκε η συνάρτηση find_text_area:

```
def find_text_area(input, output):
    ret, thresh1 = cv2.threshold(input, 0, 255, cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV)
    rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (40, 40))
    dilation = cv2.dilate(thresh1, rect_kernel, iterations = 1)
    _, contours, _ = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    #font specs
    number_box = 1
    font = cv2.FONT_HERSHEY_SIMPLEX
    fontScale = 2
    fontColor = (0,0,255)
    lineType = 2

    contours.reverse()

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)

        # Drawing a rectangle on copied image
        if cv2.contourArea(cnt) > 3000:
            rect = cv2.rectangle(output, (x, y), (x + w, y + h), (0, 0, 255), 2)
            cv2.putText(output, str(number_box),
                    (x + 20, y  + 40),
                    font,
                    fontScale,
                    fontColor,
                    lineType)
            number_box = number_box + 1

    return contours
```

Η συνάρτηση δέχεται ως παραμέτρους δύο ορίσματα. Το όρισμα input αποτελεί το σήμα της εικόνας με βάση την οποία θα υπολογιστούν οι υποπεριοχές ενώ το σήμα output, της εικόνας στην οποία θα σχεδιαστούν οι υποπεριοχές αυτές και ο αύξων αριθμός κάθε μίας.

Η συνάρτηση αρχικά εφαρμόζει ένα κατώφλι στις τιμές των εικονοστοιχείων. Συγκεκριμένα, η κατάλληλη τιμή κατωφλίου καθορίζεται από την OpenCV για την κάθε περιοχή και το αποτέλεσμα είναι μία δυαδική εικόνα λευκών και μαύρων εικονοστοιχείων. Ωστόσο αυτό που έχει κυρίως αξία στην μεθοδολογία είναι το γεγονός ότι τα εικονοστοιχεία αντιστρέφονται, δηλαδή εικονοστοιχεία που τείνουν στο λευκό παίρνουν την τιμή του μαύρου και αντίστροφα. Αυτό συμβαίνει καθώς η εύρεση των υποπεριοχών βασίζεται στον εντοπισμό των λευκών εικονοστοιχείων , συνεπώς τα γράμματα θέλουμε να απεικονίζονται λευκά και το υπόβαθρο μαύρο.

Το αποτέλεσμα της δυαδικής εικόνας:

processes, which interrelates states and maps states to observables. Finally, a reward function is associated with each automaton.

Constraint-based Trellis Diagram. Mode estimation encodes Probabilistic Hierarchical Constraint Automata (PHCA) as a constraint-based trellis diagram, and searches this diagram in order to estimate the most likely system diagnoses. This encoding is similar in spirit to a SatPlan/Graphplan encoding in planning.

Conclusion

We have extended a system capable of diagnosing and reconfiguring redundant hardware systems so that instrumented software systems can likewise be made robust. Software systems lack many of the attributes of hardware systems to which the described methods have traditionally been applied; they tend to be more hierarchical and have more complex and numerous component interactions. Software components and their interconnections represent a significantly higher modeling burden.

Our approach differs from other similar techniques in the following ways:

- Models specify program behavior in terms of abstract states, which simultaneously makes the models easier to read and think about and somewhat robust to changes in low level software implementation decisions.
- Modeling covers a wide spectrum of software considerations from a high-level storyboarding of the software to temporal considerations, if any, to the causal relationships between components.
- Robustness and recovery derives from a collection of complex and highly tuned reasoning algorithms that estimate state, choose contingencies, and plan state trajectories. The programmer is largely shielded from this complexity because the mechanism is hidden behind the intuitive unified modeling language.

An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault, incorrect diagnosis of the fault, and attributing faults to components that are operating correctly. In this sense an incorrect model is no different from any other bug in the software. It is somewhat easier to deal with, however, because the models are written at a more abstract level than the program itself, making them easier to read. There is a problem with making changes to the software definition and neglecting to update the models of the software. In time we expect tools to evolve to address this kind of problem.

The nature of the models developed for a software system vary depending upon the nature of the software itself. Some programs, especially those involved in embedded and robotic applications, have critical timing considerations that must be modeled as such whereas other programs have no timing of synchronization considerations.

Developing model-based reconfigurable software systems is a relatively new endeavor, but results of our early experiments are encouraging. Much work remains to extend the current experimental system to cover the full range of software practice.

References

1. Bernard, D., Dorais, G., Gamble, E., et al. Spacecraft autonomy flight experience: The DS1 remote agent experiment. In *Proceedings of the AIAA Space Technology Conference and Exposition* (Albuquerque, NM, Sept. 1999).
2. Firby, R. *The RAP Language Manual*. Working Note AAP-6, University of Chicago, 1995.
3. Gat, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the AAAI Fall Symposium on Plan Execution* (Cambridge, MA, Nov. 1996).
4. Laddaga, R., Robertson, P., and Shrobe, H.E. Introduction to self-adaptive software: Applications. In *Proceedings of the 2nd International Workshop on Self-Adaptive Software (IWSAS 2001)* (Balatonfured, Hungary, May 2001). LNCS 2614, Springer.
5. Mikaelian, T., Williams, B.C., and Sachenbacher, M. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05)* (Monterey, CA, June 2005).
6. Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation 10*, 1 (1994), 34–43.
7. Williams, B. and Nayak, P. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)* (Nagoya, Japan, August 1997).

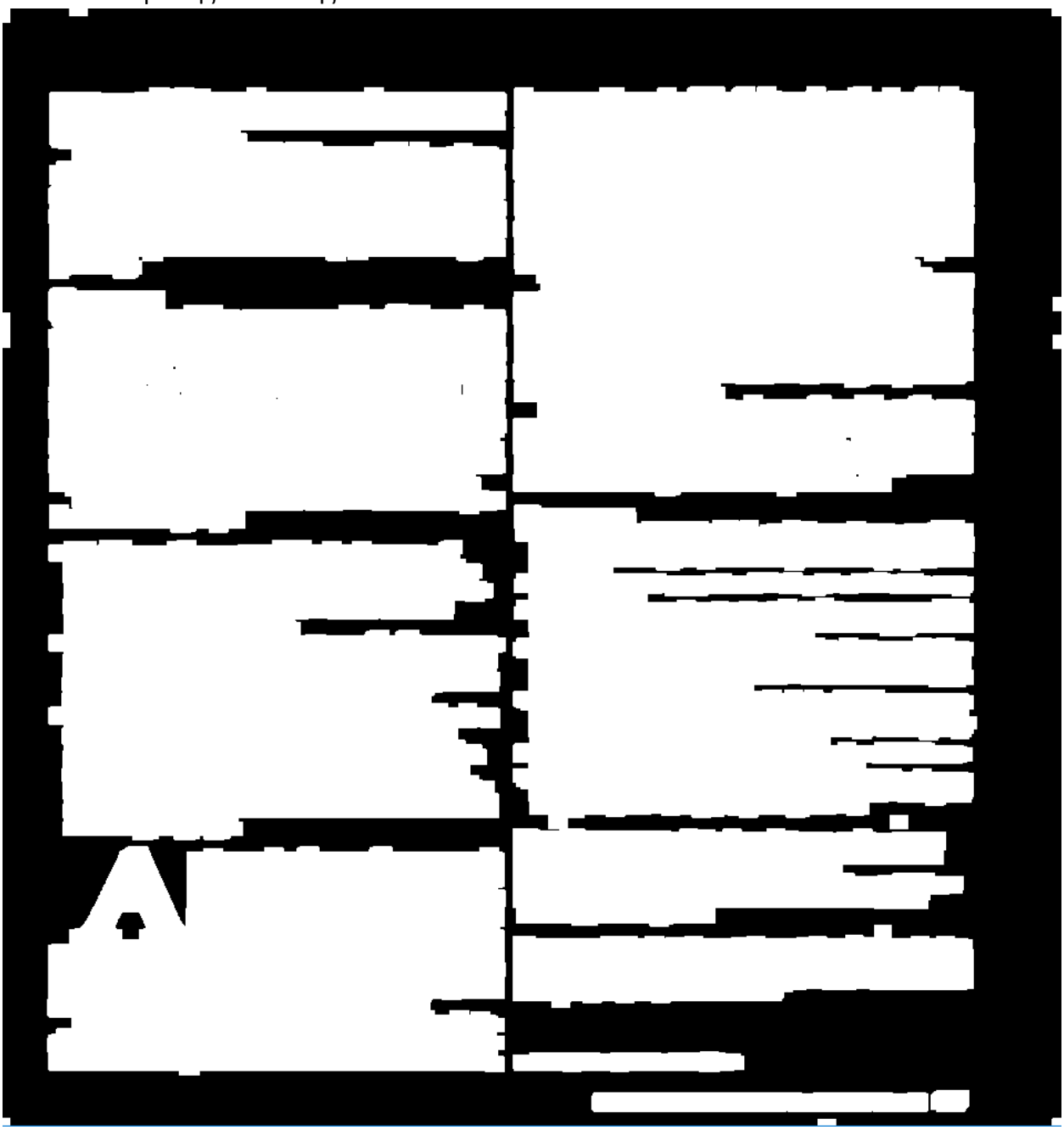
Paul Robertson (paulr@csail.mit.edu) is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.

Brian Williams (williams@mit.edu) is Boeing Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, MA.

Εικόνα 6: αρχείο «2_noise.png», μετά την εφαρμογή κατωφλίου.

Στην συνέχεια τα γράμματα διαστέλλονται ώστε οι περιοχές να αποτελούνται από ενιαία λευκά εικονοστοιχεία. Για το σκοπό αυτό δημιουργείται ένα ορθογώνιο παραλληλεπίπεδο kernel μεγέθους 40x40, το οποίο διατρέχει την εικόνα εισόδου. Αν τουλάχιστον ένα εικονοστοιχείου του kernel είναι ‘1’ (διάφορο του μηδέν) τότε αυξάνεται η περιοχή των λευκών εικονοστοιχείων . Το μέγεθος αυτό επιλέχθηκε μετά από αρκετές δοκιμές και παρατηρήθηκε ότι παράγει ικανοποιητικά αποτελέσματα για τον μετέπειτα εντοπισμό των υποπεριοχών.

Το αποτέλεσμα της διαστολής:



Εικόνα 6: αρχείο «2_noise.png», μετά την διαστολή.

Έπειτα εύκολα εντοπίζουμε το περίγραμμα των υποπεριοχών αυτών με την βοήθεια της OpenCV. Ένα πρόβλημα που δημιουργείται είναι ότι στα περιγράμματα που εντοπίζονται περιλαμβάνονται και περιοχές οι οποίες είναι άνευ ενδιαφέροντος και δημιουργούνται εξαιτίας εναπομείναντα θόρυβο ή λόγω ιδιομορφιών του εγγράφου (πχ διαχωριστικές γραμμές μεταξύ παραγράφων). Για την αντιμετώπιση του παραπάνω προβλήματος υπολογίζεται το εμβαδόν της κάθε περιοχής και αν αυτό είναι μεγαλύτερο της τιμής '3000' τότε το περίγραμμα σχεδιάζεται στην εικόνα του σήματος 'output', με τον αύξων αριθμό του. Ο παραπάνω τιμή προέκυψε πειραματικά, μετά από δοκιμές στα πέντε διαφορετικά δείγματα εγγράφων. Στο τέλος η συνάρτηση επιστρέφει τα περιγράμματα των υποπεριοχών, για την υλοποίηση της μετέπειτα μεθοδολογίας. Το αποτέλεσμα της συνάρτησης είναι:

processes, which interrelates states and maps states to observables. Finally, a reward function is associated with each automaton.

Constraint-based Trellis Diagram. Mode estimation encodes Probabilistic Hierarchical Constraint Automata (PHCA) as a constraint-based trellis diagram, and searches this diagram in order to estimate the most likely system diagnoses. This encoding is similar in spirit to a SatPlan/Graphplan encoding in planning.

Conclusion

We have extended a system capable of diagnosing and reconfiguring redundant hardware systems so that instrumented software systems can likewise be made robust. Software systems lack many of the attributes of hardware systems to which the described methods have traditionally been applied; they tend to be more hierarchical and have more complex and numerous component interactions. Software components and their interconnections represent a significantly higher modeling burden.

Our approach differs from other similar techniques in the following ways:

- Models specify program behavior in terms of abstract states, which simultaneously makes the models easier to read and think about and somewhat robust to changes in low-level software implementation decisions.
- Modeling covers a wide spectrum of software considerations from a high-level storyboarding of the software to temporal considerations, if any, to the causal relationships between components.
- Robustness and recovery derives from a collection of complex and highly tuned reasoning algorithms that estimate state, choose contingencies, and plan state trajectories. The programmer is largely shielded from this complexity because the mechanism is hidden behind the intuitive unified modeling language.

An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault, incorrect diagnosis of the fault, and attributing faults to components that are operating correctly. In this sense an incorrect model is no different from any other bug in the software. It is somewhat easier to deal with, however, because the models are written at a more abstract level than the program itself, making them easier to read. There is a problem with making changes to the software definition and neglecting to update the models of the software. In time we expect tools to evolve to address this kind of problem.

The nature of the models developed for a software system vary depending upon the nature of the software itself. Some programs, especially those involved in embedded and robotic applications, have critical timing considerations that must be modeled as such whereas other programs have no timing of synchronization considerations.

Developing model-based reconfigurable software systems is a relatively new endeavor, but results of our early experiments are encouraging. Much work remains to extend the current experimental system to cover the full range of software practice. C

References

1. Bernard, D., Dorais, G., Gamble, E., et al. Spacecraft autonomy flight experience: The DS1 remote agent experiment. In *Proceedings of the AIAA Space Technology Conference and Exposition* (Albuquerque, NM, Sept. 1999).
2. Firby, R. *The RAP Language Manual.* Working Note AAP-6, University of Chicago, 1995.
3. Gat, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the AAAI Fall Symposium on Plan Execution* (Cambridge, MA, Nov. 1996).
4. Laddaga, R., Robertson, P., and Shrobe, H.E. Introduction to self-adaptive software: Applications. In *Proceedings of the 2nd International Workshop on Self Adaptive Software (IWSAS 2001)*, (Balatonfired, Hungary, May 2001), LNCS 2614, Springer.
5. Mikaelian, T., Williams, B.C., and Sachenbacher, M. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05)*, (Monterey, CA, June 2005).
6. Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation 10*, 1 (1994), 34–43.
7. Williams, B. and Nayak, P. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, (Nagoya, Japan, August 1997).

Paul Robertson (paulr@csail.mit.edu) is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.

Brian Williams (williams@mit.edu) is Boeing Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, MA.

Επεξεργασία υποπεριοχών

Εύρεση επιφάνειας κειμένου

Στην συνέχεια της μεθοδολογίας υπολογίζεται η επιφάνεια της υποπεριοχής που καταλαμβάνεται από κείμενο, η επιφάνεια του περιβάλλοντος κουτιού, ο αριθμός των λέξεων της υποπεριοχής και η μέση τιμή διαβάθμισης του γκρι.

Για την εύρεση της επιφάνειας της υποπεριοχής που καταλαμβάνεται από κείμενο, χρησιμοποιείται η συνάρτηση calc_text_area, η οποία αξιοποιεί την συνάρτηση connectedComponents της OpenCV.

```
def calc_text_area(img, min_size=15):
    # find all your connected components (white blobs in your image)
    nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(img, connectivity=8)
    # connectedComponentswithStats yields every seperated component with information on each of them, such as size
    # the following part is just taking out the background which is also considered a component, but most of the time we don't want that.
    sizes = stats[1:, -1]
    nb_components = nb_components - 1
    pixel_size = 0
    # for every component in the image, you keep it only if it's above min_size
    for i in range(0, nb_components):
        if (sizes[i] < min_size):
            img[output == i + 1] = 0
        else:
            pixel_size += sizes[i]

    print(pixel_size)
```

Η παραπάνω συνάρτηση δέχεται ως ορίσματα δύο παραμέτρους την img, η οποία αντιπροσωπεύει το σήμα της υποπεριοχής της δυαδικής εικόνας (βλέπε ενότητα «Εντοπισμός υποπεριοχών») και την min_size η οποία αρχικοποιείται στην τιμή 15 και χρησιμοποιείται για τον έλεγχο μεγέθους επιφάνειας (θα γίνει αντιληπτή η χρησιμότητα της στην συνέχεια). Με χρήση της συνάρτηση connectedComponentsWithStats της OpenCV εντοπίζουμε όλα τα "αντικείμενα" της υποπεριοχής τα οποία αποτελούνται από λευκά εικονοστοιχεία. Θυμίζετε ότι στην δυαδική εικόνα το κείμενο χαρακτηρίζεται από τα λευκά εικονοστοιχεία. Από τα “αντικείμενα” όσα έχουν επιφάνεια μικρότερη της παραμέτρου min_size (πιθανός εναπομένοντας θόρυβος) διαγράφονται και για τα υπόλοιπα υπολογίζεται η επιφάνεια τους. Η τιμή min_size = 15 επιλέχθηκε μετά από δοκιμές και οδηγεί σε αποτελέσματα με ελαχίστη ή τις περισσότερες φορές μηδαμινή απώλεια πληροφορίας.

Ένα πρόβλημα που προκύπτει από τον παραπάνω υπολογισμό είναι ότι κάποιες υποπεριοχές περιέχουν διαχωριστικές γραμμές, πλαίσια, σύμβολα στοιχειοθέτησης (κουκκίδες) και άλλα

ειδικά στοιχειά, η περιοχή των οποίων υπολογίζεται στο τελικό αποτέλεσμα της επιφάνειας. Μια πρώτη σκέψη για την αντιμετώπιση αυτού του προβλήματος θα ήταν να αυξηθεί ο παράγοντας min_size. Όμως κάτι τέτοιο παράγει μη αποδεκτά αποτελέσματα καθώς τις περισσότερες φόρες υπάρχει μεγάλη απώλεια πληροφορίας. Μια πιο επιτυχημένη λύση θα ήταν να υπολογιστούν οι υποπεριοχές λέξεων ή ακόμη και των ξεχωριστών γραμμάτων και με βάση αυτές να βρεθούν τα αντικείμενα τα οποία της αποτελούν. Η παρούσα μεθοδολογία προσφέρει αυτή την δυνατότητα με μερικές τροποποιήσεις, αλλά για λόγους απλότητας δεν υλοποιήθηκε.

Υπολογισμός επιφάνειας περιβάλλοντος κουτιού

Αφού έχουμε υπολογίσει προηγούμενος τα περιβάλλοντα κουτιά των υποπεριοχών (βλέπε ενότητα Εντοπισμός υποπεριοχών), μπορούμε εύκολα να βρούμε την επιφάνεια τους χρησιμοποιώντας την συνάρτηση contourArea της OpenCV.

```
cv2.contourArea(cnt)
```

Όπου cnt το περίγραμμα της κάθε υποπεριοχής.

Υπολογισμός αριθμός λέξεων υποπεριοχής

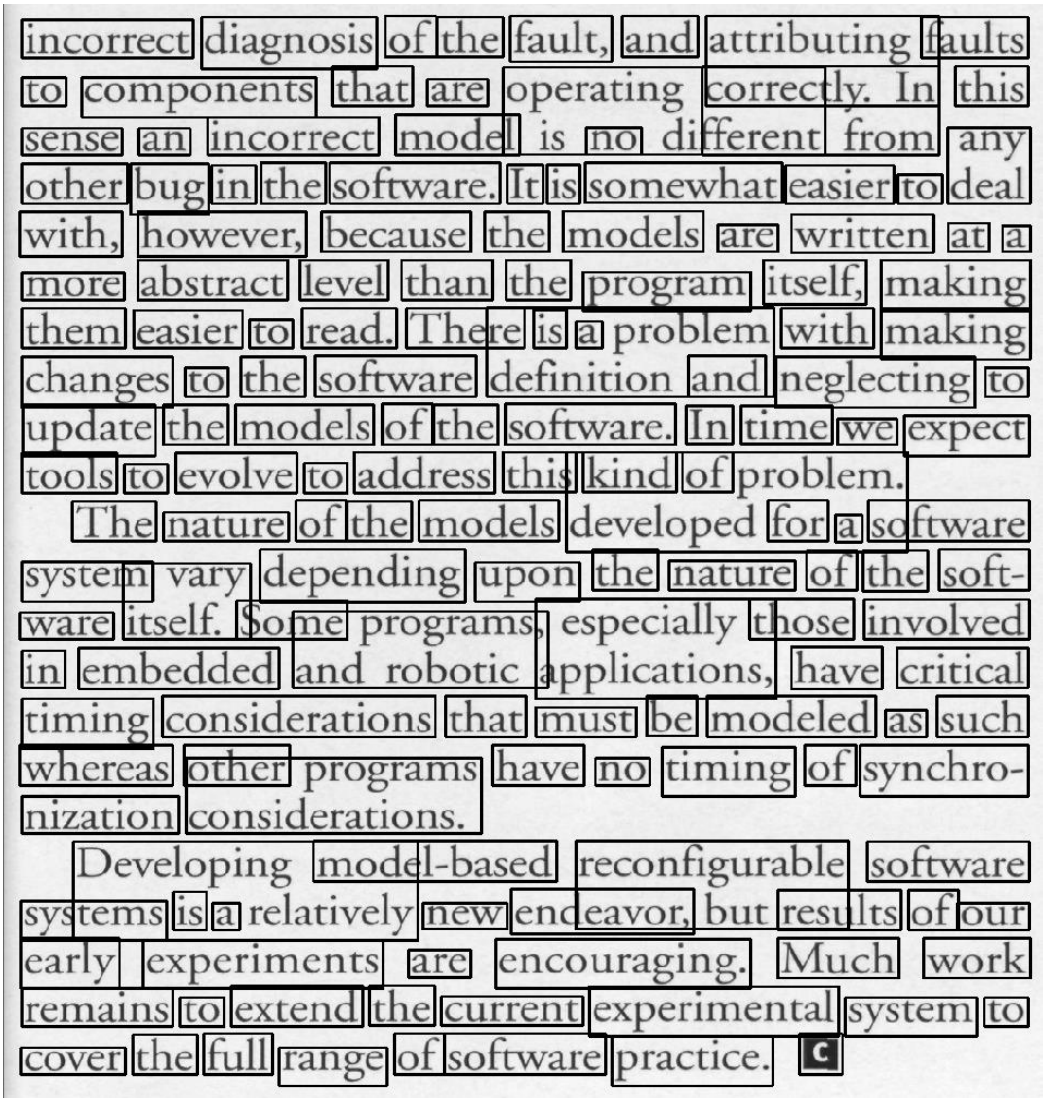
Για την εύρεση του αριθμού των λέξεων της υποπεριοχής θα χρησιμοποιηθεί η ίδια μεθοδολογία με τον εντοπισμό των υποπεριοχών. Η συνάρτηση που υλοποιεί την παραπάνω μεθοδολογία για την εύρεση των λέξεων είναι η word_count

```
def word_count(img):

    rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (7, 7))
    dilation = cv2.dilate(img, rect_kernel, iterations=1)

    _, contours, _ = cv2.findContours(dilation, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    words = 0

    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt)

        # Drawing a rectangle on copied image
        area = cv2.contourArea(cnt)
        if area > 200:
            words += 1
            rect = cv2.rectangle(croppedImg, (x, y), (x + w, y + h), (0, 0, 255), 2)
    print(words)
```

Η μοναδική παράμετρος img της συνάρτησης, χαρακτηρίζει το σήμα το δυαδικής εικόνας της υποπεριοχής, όπως έχει υπολογιστεί παραπάνω. Η μόνη διαφοροποίηση με την μεθοδολογία εντοπισμού των υποπεριοχών είναι ότι τώρα εντοπίζουμε την περιοχή της κάθε λέξης, συνεπώς για το σκοπό απαιτείται λιγότερο έντονη διαστολή στα γράμματος. Έτσι, το μέγεθος του kernel επιλέγεται να είναι 7x7. Η επιλογή αύτη σε συνδυασμό με την αφαίρεση όσων περιοχών έχουν επιφάνεια μικρότερη από την τιμή 200 δίνει ικανοποιείται αποτελέσματα

Παρατίθενται δείγματα εντοπισμού των λέξεων (έχουν σχεδιαστεί τα περιγράμματα γύρω από τις περιοχές των λέξεων ώστε να γίνει ο έλεγχος της μεθόδου):

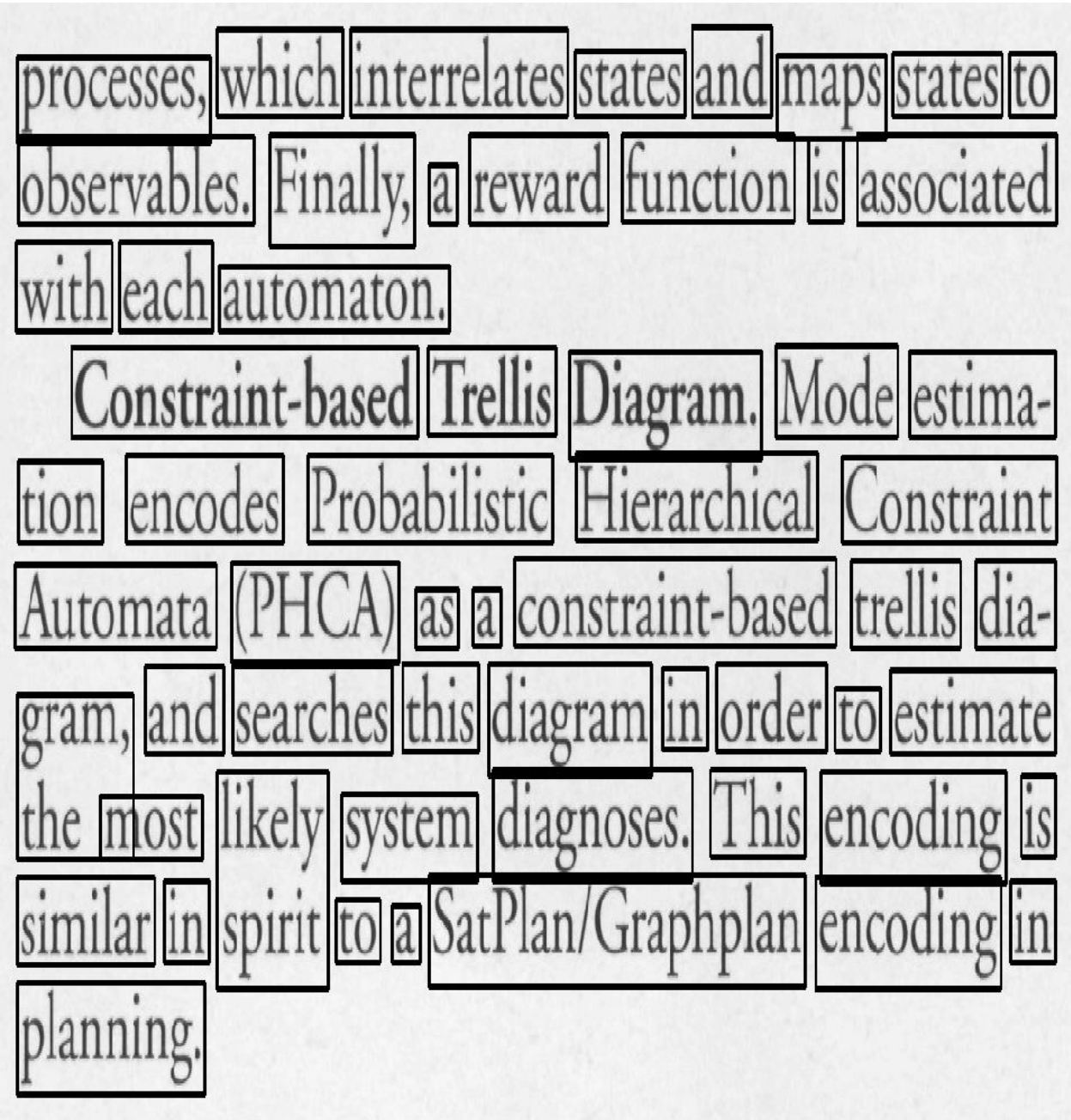
1) Δείγμα



Όπως γίνεται εύκολα αντιληπτό υπάρχουν περιοχές, οι οποίες περιλαμβάνουν περισσότερες από μία λέξεις. Έγινε προσπάθεια να αντιμετωπιστεί το πρόβλημα αυτό μειώνοντας το μέγεθος του kernel κατά την διαστολή, αλλά η βελτίωση δεν ήταν σημαντική και επιπλέον δημιουργούνταν το πρόβλημα, ότι μερικές λέξεις έσπαζαν σε περισσότερες από μία περιοχές.

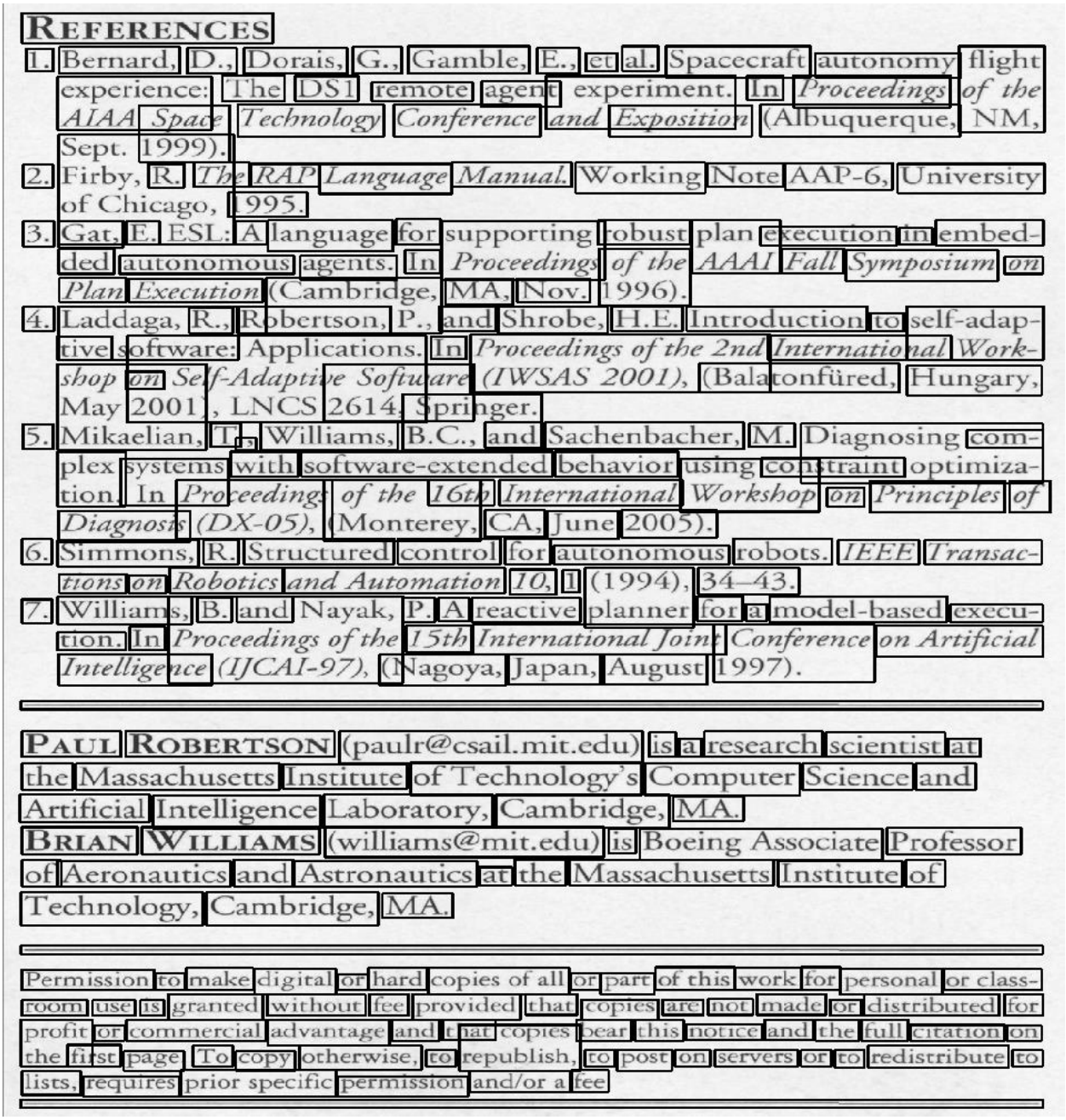
Η λέξεις της παραπάνω υποπεριοχής είναι 171 ενώ με την μεθοδολογία μετρήθηκαν 167, δηλαδή μία απόκλιση της τάξης του 2.339%.

2) Δείγμα



Εδώ οι μεθοδολογία πετυχαίνει λίγο χειρότερα αποτελέσματα. Καταφέρνει να εντοπίσει συνολικά 60 λέξεις από τις 62, δηλαδή μία απόκλιση 3.22%. Το παραπάνω παράδειγμα επίσης παρουσιάζει μία ακόμη αδυναμία της μεθοδολογίας. Λέξεις που διακόπτονται λόγω της εναλλαγής γραμμής, διαιρούνται σε δύο περιοχές και λέξεις όπως «constraint-based», οι οποίες θεωρούνται ως δύο διακριτές λέξεις, κατά τον εντοπισμό τους υπολογίζονται ως μία.

3) Δείγμα



REFERENCES

1. Bernard, D., Dorais, G., Gamble, E., et al. Spacecraft autonomy flight experience: The DS1 remote agent experiment. In *Proceedings of the AIAA Space Technology Conference and Exposition* (Albuquerque, NM, Sept. 1999).
2. Firby, R. *The RAP Language Manual.* Working Note AAP-6, University of Chicago, 1995.
3. Gat, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the AAAI Fall Symposium on Plan Execution* (Cambridge, MA, Nov. 1996).
4. Laddaga, R., Robertson, P., and Shrobe, H.E. Introduction to self-adaptive software: Applications. In *Proceedings of the 2nd International Workshop on Self-Adaptive Software (IWSAS 2001)*, (Balatonfüred, Hungary, May 2001), LNCS 2614, Springer.
5. Mikaelian, T., Williams, B.C., and Sachenbacher, M. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05)*, (Monterey, CA, June 2005).
6. Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation 10*, 1 (1994), 34–43.
7. Williams, B. and Nayak, P. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, (Nagoya, Japan, August 1997).

PAUL ROBERTSON (paulr@csail.mit.edu) is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.

BRIAN WILLIAMS (williams@mit.edu) is Boeing Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, MA.

Το δείγμα αυτό είναι ιδιαίτερα ιδιόμορφο και η ακρίβεια μειώθηκε, με τον εντοπισμό 292 λέξεων από τις 314, δηλαδή 7% απόκλιση. Στο δείγμα αυτό εμφανίζεται πάλι το πρόβλημα το διαχωριστικών γραμμών, οι οποίες μετρούνται ως λέξη. Είχε γίνει δοκιμή να χρησιμοποιηθεί άνω όριο στις επιφάνειες των περιοχών των λέξεων, ώστε περιοχές όπως αυτές των διαχωριστικών γραμμών να μην υπολογίζονται. Ωστόσο έγινε αντιληπτό ότι η επιφάνεια αυτών των γραμμών, λόγω του περιορισμένου τους πάχους, είναι μικρότερη από την επιφάνεια περιοχών αρκετών λέξεων με μεσαίο η μεγάλο μήκος. Με αποτέλεσμα να αποκλείονται κι εκείνες από την μέτρηση.

Τα αποτελέσματα της μεθοδολογίας φαίνονται αρκετά ικανοποιητικά. Ίσως να ήταν δυνατή μία πιο ακριβής καταμέτρησε με μεθόδους OPC (Optical Character Recognition) .

Υπολογισμός μέσης τιμής διαβάθμισης του γκρι στην υποπεριοχή

Ο υπολογισμός της μέσης τιμής διαβάθμισης του γκρι σε κάθε υποπεριοχή, η εκτέλεση του οποίου να είναι ανεξάρτητη του μεγέθους της θα υλοποιηθεί με την μέθοδο που βασίζεται στην εικόνα ολοκλήρωμα. Αφού έχει δημιουργηθεί η εικόνα ολοκλήρωμα , με βάση τις διαστάσεις τις κάθε υποπεριοχής και τον παρακάτω τύπο, μπορεί να επιτευχθεί ο υπολογισμός.

$$I(x,y) = i(x,y) + I(x,y-1) + I(x-1,y) - I(x-1,y-1)$$

Η συνάρτηση που δημιουργήθηκε για τον υπολογισμό της μέσης τιμής του γκρι είναι η integral_gray:

```
def integral_gray(int_img, x ,y , w ,h):

    img_area = int_img[y + h - 1, x + w - 1] + int_img[y - 1, x  - 1] - int_img[y + h - 1, x - 1 ] - int_img[y - 1, x + w - 1]
    gray_mean = img_area / (w*h)
    print("Gray Mean", gray_mean)
```

Η συνάρτηση δέχεται σαν όρισμα την εικόνα ολοκλήρωμα, η οποία υπολογίζεται από την συνάρτηση της OpenCV integral( ), τις συντεταγμένες x και y της υποπεριοχής και τις διαστάσεις αυτής. Τα αποτελέσματα της μεθόδου, αντιστοιχούν με αυτά του συνήθη τρόπου εύρεσης μέσης τιμής, ο οποίος βασίζεται στην άθροιση όλων των στοιχείων και την διαίρεση με το πλήθος αυτών. Η τιμές που προέκυψαν παρατίθενται συνολικά στη επόμενη ενότητα.

Χρήση της μεθοδολογίας

Για την χρήση της μεθοδολογίας που αναπτύχθηκε δημιουργήθηκε η συνάρτηση hw, η οποία συνδυάζει όλα τα παραπάνω βήματα.

```
def hw1(original, filename):

    original = cv2.imread(original)
    image = cv2.imread(filename)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    img1 = gray.copy()
    median_filter(gray, img1)
    thresh, contours = find_text_area(img1, original)
    intergral_image = cv2.integral(img1)

    for cnt in  contours:
        i = 1
        x, y, w, h = cv2.boundingRect(cnt)
        cropped = thresh[y:y + h, x:x + w]
        if cv2.contourArea(cnt) > 3000:
            print("---- Region ", i, ": ----")
            calc_text_area(cropped)
            print("Bounding Box Area (px): ", cv2.contourArea(cnt))
            word_count(cropped)
            integral_gray(intergral_image, x, y, w, h)

    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.imshow('image', original)
    cv2.waitKey(0)
```

Η συνάρτηση δέχεται δύο ορίσματα, το original το οποίο είναι το σήμα της εικόνας χωρίς τον θόρυβο (ώστε να σχεδιαστούν τα περιγράμματα των υποπεριοχών σε αυτήν) και το όρισμα filename το οποίο περιλαμβάνει την εικόνα προς επεξεργασία. Αν δεν διατίθενται δύο εκδοχές την εικόνας μπορεί δοθεί η ίδια τιμή και στις δύο παραμέτρους.

Η συνάρτηση διαβάζει τις εικόνες των παραμέτρων και μετατρέπει αυτή προς επεξεργασία από εικόνα τριών καναλιών σε ένα. Στην συνέχεια εφαρμόζεται το Median φίλτρο, υπολογίζονται οι υποπεριοχές και η εικόνα ολοκλήρωμα. Τέλος οι υποπεριοχές επεξεργάζονται για υπολογιστεί η επιφάνεια τους, η επιφάνεια κείμενου, ο αριθμός των λέξεων και μέση τιμή διαβάθμισης του γκρι.

Παρακάτω παρατίθεται το τελικό αποτέλεσμα:

processes, which interrelates states and maps states to observables. Finally, a reward function is associated with each automaton.

Constraint-based Trellis Diagram. Mode estimation encodes Probabilistic Hierarchical Constraint Automata (PHCA) as a constraint-based trellis diagram, and searches this diagram in order to estimate the most likely system diagnoses. This encoding is similar in spirit to a SatPlan/Graphplan encoding in planning.

Conclusion

We have extended a system capable of diagnosing and reconfiguring redundant hardware systems so that instrumented software systems can likewise be made robust. Software systems lack many of the attributes of hardware systems to which the described methods have traditionally been applied; they tend to be more hierarchical and have more complex and numerous component interactions. Software components and their interconnections represent a significantly higher modeling burden.

Our approach differs from other similar techniques in the following ways:

- Models specify program behavior in terms of abstract states, which simultaneously makes the models easier to read and think about and somewhat robust to changes in low-level software implementation decisions.
- Modeling covers a wide spectrum of software considerations from a high-level storyboarding of the software to temporal considerations, if any, to the causal relationships between components.
- Robustness and recovery derives from a collection of complex and highly tuned reasoning algorithms that estimate state, choose contingencies, and plan state trajectories. The programmer is largely shielded from this complexity because the mechanism is hidden behind the intuitive unified modeling language.

An interesting feature of our approach is the ability to add robustness incrementally. More modeling leads to greater runtime robustness because it allows the system to detect, diagnose, and repair more fault situations. This means the effort devoted to modeling can be managed in much the same way that is done for test suite development in conventional software development projects.

Modeling errors can result in a number of undesirable outcomes such as failure to detect fault conditions and subsequent failure to recover from the fault, incorrect diagnosis of the fault, and attributing faults to components that are operating correctly. In this sense an incorrect model is no different from any other bug in the software. It is somewhat easier to deal with, however, because the models are written at a more abstract level than the program itself, making them easier to read. There is a problem with making changes to the software definition and neglecting to update the models of the software. In time we expect tools to evolve to address this kind of problem.

The nature of the models developed for a software system vary depending upon the nature of the software itself. Some programs, especially those involved in embedded and robotic applications, have critical timing considerations that must be modeled as such whereas other programs have no timing of synchronization considerations.

Developing model-based reconfigurable software systems is a relatively new endeavor, but results of our early experiments are encouraging. Much work remains to extend the current experimental system to cover the full range of software practice. ■

References

1. Bernard, D., Dorais, G., Gamble, E., et al. Spacecraft autonomy flight experience: The DS1 remote agent experiment. In *Proceedings of the AIAA Space Technology Conference and Exposition* (Albuquerque, NM, Sept. 1999).
2. Firby, R. *The RAP Language Manual.* Working Note AAP-6, University of Chicago, 1995.
3. Gat, E. ESL: A language for supporting robust plan execution in embedded autonomous agents. In *Proceedings of the AAAI Fall Symposium on Plan Execution* (Cambridge, MA, Nov. 1996).
4. Laddaga, R., Robertson, P., and Shrobe, H.E. Introduction to self-adaptive software: Applications. In *Proceedings of the 2nd International Workshop on Self-Adaptive Software (IWSAS 2001)*, (Balatonfüred, Hungary, May 2001), LNCS 2614, Springer.
5. Mikaelian, T., Williams, B.C., and Sachenbacher, M. Diagnosing complex systems with software-extended behavior using constraint optimization. In *Proceedings of the 16th International Workshop on Principles of Diagnosis (DX-05)*, (Monterey, CA, June 2005).
6. Simmons, R. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation 10*, 1 (1994), 34–43.
7. Williams, B. and Nayak, P. A reactive planner for a model-based executive. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, (Nagoya, Japan, August 1997).

Paul Robertson (paulr@csail.mit.edu) is a research scientist at the Massachusetts Institute of Technology's Computer Science and Artificial Intelligence Laboratory, Cambridge, MA.
Brian Williams (williams@mit.edu) is Boeing Associate Professor of Aeronautics and Astronautics at the Massachusetts Institute of Technology, Cambridge, MA.

---- Region 1 : ----

Area (px): 151558

Bounding Box Area (px): 1136012.5

Number of words: 180

Mean gray-level value in bounding box: 218.64882172813208

---- Region 2 : ----

Area (px): 62261

Bounding Box Area (px): 469824.0

Number of words: 71

Mean gray-level value in bounding box: 220.7881738346897

---- Region 3 : ----

Area (px): 75732

Bounding Box Area (px): 620917.0

Number of words: 119

Mean gray-level value in bounding box: 223.06973647440196

---- Region 4 : ----

Area (px): 102245

Bounding Box Area (px): 1022821.5

Number of words: 343

Mean gray-level value in bounding box: 228.40319790859456

---- Region 5 : ----

Area (px): 89086

Bounding Box Area (px): 743121.5

Number of words: 156

Mean gray-level value in bounding box: 224.72709608638834

---- Region 6 : ----

Area (px): 80864

Bounding Box Area (px): 592606.0

Number of words: 99

Mean gray-level value in bounding box: 222.3688143045053

---- Region 7 : ----

Area (px): 12120

Bounding Box Area (px): 176010.0

Number of words: 87

Mean gray-level value in bounding box: 229.73373602892835

---- Region 8 : ----

Area (px): 1156

Bounding Box Area (px): 26866.0

Number of words: 10

Mean gray-level value in bounding box: 232.05061390631255

---- Region 9 : ----

Area (px): 630

Bounding Box Area (px): 5102.0

Number of words: 1

Mean gray-level value in bounding box: 222.35493212669684

---- Region 10 : ----

Area (px): 5726

Bounding Box Area (px): 40751.0

Number of words: 9

Mean gray-level value in bounding box: 218.48467432950193