1. ΓΕΝΙΚΗ ΑΝΑΛΥΣΗ ΠΡΟΒΛΗΜΑΤΟΣ

Στα πλαίσια της ομάδα του 2012-2013 αποφασίστηκε η χρήση στερεοσκοπικής κάμερας, η οποία σε συνδυασμό με την απλή (monocular) κάμερα θα εντόπιζε τρύπες στους τοίχους της πίστας του διαγωνισμού. Ο λόγος που επιλέξαμε να χρησιμοποιήσουμε στερεοσκοπική κάμερα εντοπίζεται στο γεγονός ότι η συγκεκριμένη κάμερα μας παρέχει πληροφορίες για την τρισδιάστατη δομή της σκηνής που βλέπουν οι κάμερες (depth analysis). Γενικά η θέση ενός σημείου στο χώρο, έστω X= (X,Y,Z) σχετίζεται με την αντίστοιχη θέση του σημείου στην εικόνα x= (x , y) με τις εξισώσεις.

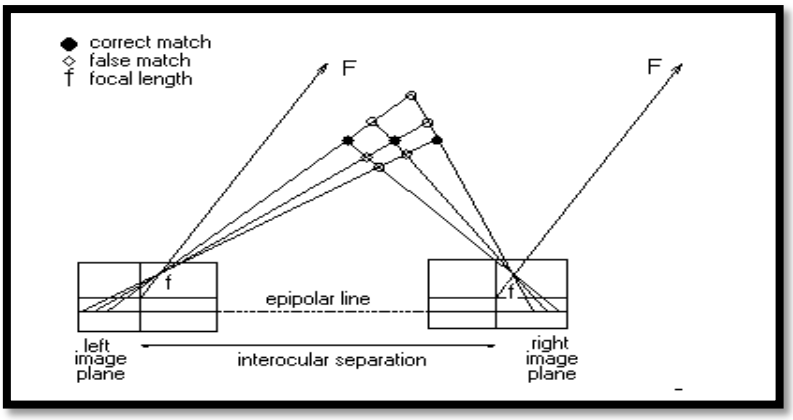
$$x = x_0 - s_{xy}c \cdot \frac{r_{11}(X - X_0) + r_{21}(Y - Y_0) + r_{31}(Z - Z_0)}{r_{13}(X - X_0) + r_{23}(Y - Y_0) + r_{33}(Z - Z_0)} + \Xi_x(\mathbf{x}) \quad (1)$$

$$y = y_0 - c \cdot \frac{r_{12}(X - X_0) + r_{22}(Y - Y_0) + r_{32}(Z - Z_0)}{r_{13}(X - X_0) + r_{23}(Y - Y_0) + r_{33}(Z - Z_0)} + \Xi_y(\mathbf{x}) \quad (2)$$

, όπου x0=(x0,x0) αναφέρεται τις συντεταγμένες της εικόνας του principal point της κάμερας, το c στο focal length της κάμερας, το X0= (X0,Y0,Z0) στις τρισδιάστατες συντεταγμένες του optical center της κάμερας και τα rij στις συνιστώσες του 3x3 rotation matrix R, ο οποίος περιγράφει τον προσανατολισμό της κάμερας.

Όπως προανέφερα στη στερεοσκοπική όραση αυτό που θέλουμε να πετύχουμε είναι η αίσθηση του βάθους. Συνεπώς χρησιμοποιώντας 2 κάμερες, οι οποίες είναι τοποθετημένες σε γνωστή απόσταση μεταξύ τους μπορούμε να μετράμε βάθος. Για να επιτύχουμε το παραπάνω πρέπει να αρχικά να επιλυθεί το correspondence problem.

Correspondence problem: Το πρόβλημα της εξακρίβωσης των σημείων μιας εικόνας που αντιστοιχούν στα σημεία μιας άλλης εικόνας. Πιο αναλυτικά αν έχουμε δυο εικόνες της ίδιας σκηνής, τραβηγμένες από διαφορετικές οπτικές γωνίες, το συγκεκριμένο πρόβλημα μεταφράζεται σε πρόβλημα εύρεσης ενός συνόλου σημείων τα οποία μπορούν να αναγνωριστούν ως ίδια σημεία στην άλλη εικόνα. Στο παρακάτω σχήμα φαίνεται γραφικά το πρόβλημα που καλούμαστε να επιλύσουμε.

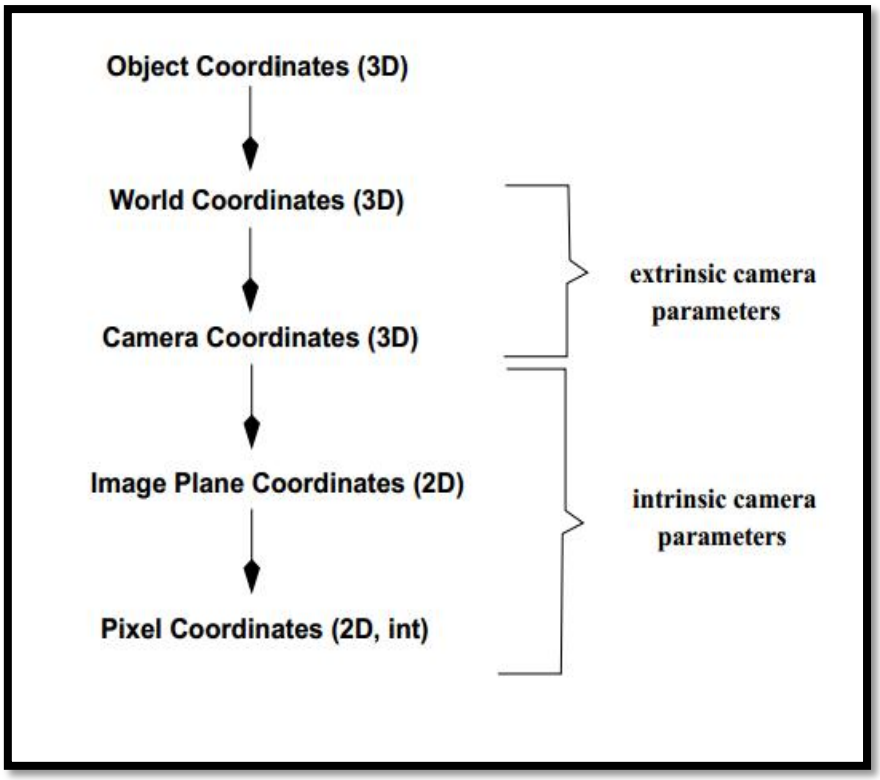


Ωστόσο για να μπορέσουμε να αξιοποιήσουμε τις πληροφορίες από την στερεοσκοπική κάμερα είναι αναγκαίο να προηγηθεί η διαδικασία του calibration. Στόχος της συγκεκριμένες διαδικασίας είναι η εξαγωγή των εσωτερικών και εξωτερικών παραμέτρων της κάμερας.

1.1 Calibration

Όπως ανέφερα προηγούμενα στόχος της συγκεκριμένης διαδικασίας είναι η εξαγωγή των εσωτερικών (intrinsic) και εξωτερικών (extrinsic) παραμέτρων της κάμερας. Συγκεκριμένα ορισμένες από αυτές θα τις χρησιμοποιήσουμε και στην συνέχεια για την εξαγωγή του point cloud και του disparity map.

Συνοπτικά αναφέρω ότι ως εξωτερικές ορίζονται εκείνες οι παράμετροι που ορίζουν τη θέση και τον προσανατολισμό της κάμερας σε σχέση με τον «κόσμο». Ουσιαστικά πρόκειται για ένα μετασχηματισμό από το σύστημα αναφοράς που ορίζει η κάμερα (camera reference frame) στο παγκόσμιο σύστημα αναφοράς (world reference frame). Οι εσωτερικές παράμετροι από την άλλη είναι απαραίτητες για τον συσχετισμό των συντεταγμένων pixel (pixel coordinates) ενός σημείου πάνω στην εικόνα με το σύστημα αναφοράς που ορίζει η κάμερα (camera reference frame). Τα παραπάνω συνοψίζονται στην παρακάτω εικόνα.



1.1.1. Εσωτερικές Παράμετροι Κάμερας (Intrinsic Parameters):

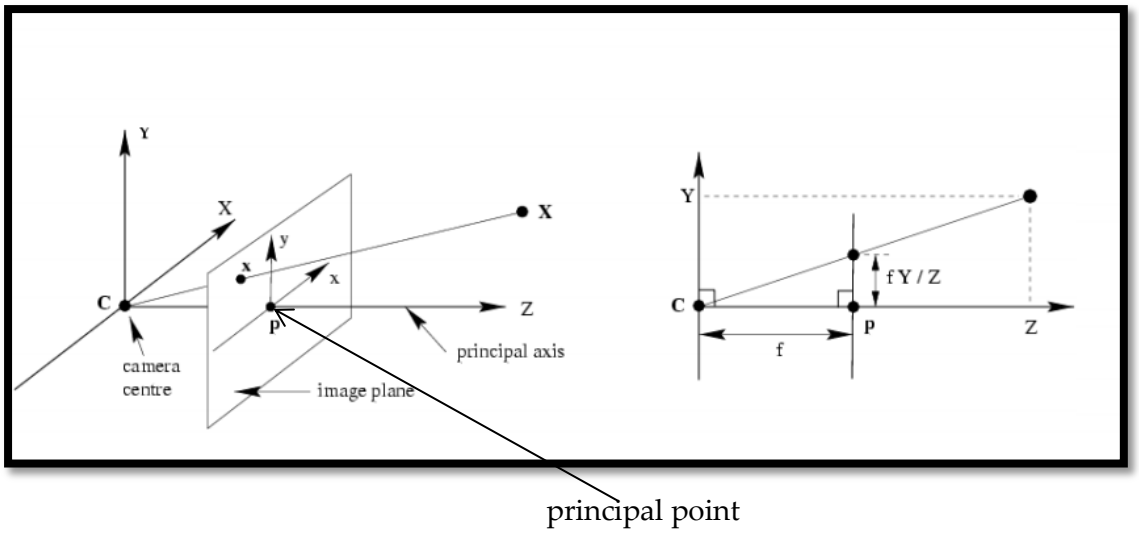
Οι εσωτερικές παράμετροι της κάμερας εξαρτώνται μόνο από τα χαρακτηριστικά της κάμερας. Αναλυτικά υπάρχουν **5 intrinsic parameters**:

- **Focal length (εστιακό μήκος):** Η εστίαση μιας κάμερας γενικά εξαρτάται από το εστιακό μήκος f. Πρόκειται για την απόσταση μεταξύ του CCD και του φακού. Επιπρόσθετα η γωνία λήψης είναι ανάλογη με το εστιακό μήκος. Συγκεκριμένα έχουμε τις τιμές του εστιακού μήκους στις δυο κατευθύνεις $\vec{x}$ και $\vec{y}$.

- **Μέγεθος pixel**: Το μέγεθος ενός pixel στις κατευθύνσεις των $\vec{x}$ και $\vec{y}$ (sx,sy). Συνήθως λαμβάνουμε αυτές τις δυο τιμές ίσες μεταξύ τους, πράγμα που σημαίνει ότι ουσιαστικά έχουμε 4 εσωτερικές παραμέτρους.

- **Principal point**: Ως principal point ορίζεται το σημείο τομής του κύριου άξονα με το πλάνο της εικόνας. Συγκεκριμένα εμείς ενδιαφερόμαστε για τις συντεταγμένες του principal point της κάμερας cx,cy.

Όλα τα προαναφερθέντα μεγέθη παρουσιάζονται γραφικά στο ακόλουθο σχήμα:



Συνεπώς με βάση όλα τα παραπάνω σχηματίζεται ο πίνακας των εσωτερικών παραμέτρων της κάμερας, ο οποίο είναι ο ακόλουθος:

$$A=\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

,όπου τα fx, fy αναφέρονται στο εστιακό βάθος (focal length) εκφρασμένο σε μονάδες pixel στις δυο κατευθύνσεις και τα cx, cy σχετίζονται με το principal point της κάμερας, το οποίο συνήθως λαμβάνεται στο κέντρο της εικόνας.

Να σημειώσω σε αυτό το σημείο ότι ισχύει $fx=\frac{f}{sx}$ και $fy=\frac{f}{sy}$.
Από όλα τα παραπάνω γίνεται εύκολα αντιληπτό ότι οι εσωτερικές παράμετροι της κάμερας, δεν εξαρτώνται από την σκηνή που βλέπει η κάμερα.

1.1.2. Εξωτερικές Παράμετροι Κάμερας (Extrinsic Parameters):

Οι εξωτερικές παράμετροι της κάμερας εξαρτώνται μόνο από τη θέση της κάμερας και ουσιαστικά πρόκειται για τη θέση και τον προσανατολισμό της στερεοσκοπικής κάμερας στο παγκόσμιο σύστημα αναφοράς. Οι εξωτερικές παράμετροι είναι αναγκαίες για την επίλυση του correspondence problem καθώς και για triangulation. Αναλυτικά είναι:

- Πίνακας στροφής R: Ο πίνακας στροφής καθορίζεται από τις 3 γωνίες στροφής γύρω από τους άξονες της δεξιάς κάμερας ($\hat{\varphi}$, $\hat{\theta}$, $\hat{\gamma}$) .
  Συγκεκριμένα η στροφή γύρω από τον άξονα των X ($\hat{\varphi}$) μπορεί να βρεθεί εξετάζοντας την ομοιομορφία του disparity map. Πιο αναλυτικά καθώς ο matching algorithm προσπαθεί να αντιστοιχίσει σημεία που βρίσκονται στην ίδια γραμμή στις δυο εικόνες, όταν οι εικόνες δεν είναι κατακόρυφα στοιχισμένες ($\hat{\varphi}$!=0) προκύπτει θόρυβος στον disparity map. Κατά αντιστοιχία υπολογίζεται και η γωνία $\hat{\gamma}$.

- Διάνυσμα μετατόπισης T : Πρόκειται για ένα μέτρο της απόστασης μεταξύ των κέντρων των δυο καμερών.

Ο συνδυασμός εξωτερικών και εσωτερικών παραμέτρων μιας κάμερας οδηγεί στον παρακάτω μετασχηματισμό, ο οποίος μας επιτρέπει να μετασχηματίσουμε συντεταγμένες σε pixel σε συντεταγμένες στον πραγματικό κόσμο.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

όπου:

- (X,Y,Z) είναι οι συντεταγμένες ενός τρισδιάστατου σημείου στο χώρο.
- (u ,v) είναι οι συντεταγμένες της προβολής του σημείου σε pixel.
- Ο πίνακας $\begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$ είναι ο πίνακας των εσωτερικών παραμέτρων της κάμερας.
- Ο πίνακας $\begin{bmatrix} r11 & r12 & r13 & t1 \\ r21 & r22 & r23 & t2 \\ r31 & r32 & r33 & t3 \end{bmatrix}$ είναι ο πίνακας των εξωτερικών παραμέτρων.

Επιπλέον με τις παραμέτρους που έχουν προκύψει από το calibration των δυο καμερών μπορούμε να υπολογίσουμε δύο άλλες πολύ σημαντικές παραμέτρους τον fundamental και τον essential matrix.

Ο essential και ο fundamental matrix είναι 3x3 πίνακες που «κωδικοποιούν» την epipolar geometry μεταξύ των δυο εικόνων. Στόχος μας είναι δοθέντος ενός σημείου στην εικόνα της μιας κάμερας και πολλαπλασιάζοντας το με τον essential και τον fundamental matrix να βρούμε ποία epipolar line να αναζητήσουμε στην εικόνα της δεύτερης.

Ο essential matrix έχει βαθμό 2 και εξαρτάται μόνο από τις εξωτερικές παραμέτρους της κάμερας και ισούται με E=R*T. Στην ουσία δηλαδή ο essential matrix μας δείχνει τον σχετικό προσανατολισμό των δυο καμερών. Ο fundamental matrix από την άλλη χρησιμοποιεί συγχρόνως τις εξωτερικές και τις εσωτερικές παραμέτρους των καμερών. Ομοίως με τον essential έτσι και αυτός είναι βαθμού 2. Σε αυτό το σημείο θα ήθελα τέλος να προσθέσω ότι για τον υπολογισμό των δυο παραπάνω πινάκων χρησιμοποιήθηκε ο Eight-Point Alogirthm.

2. ΑΝΑΛΥΣΗ ΜΕΘΟΔΟΛΟΓΙΑΣ

2.1. Γενική προσέγγιση μεθοδολογίας για την εύρεση τρύπας

Η λογική που επιλέξαμε να ακολουθήσουμε για τον εντοπισμό τρύπας παρουσιάζει σημαντικές ομοιότητες με την λογική στην περίπτωση της monocular camera. Συγκεκριμένα σε πρώτο στάδιο προσπαθούμε να εντοπίσουμε περιοχές ενδιαφέροντος στην εικόνα που βλέπουν οι δυο κάμερας και στη συνέχεια συμπεραίνουμε αν είναι τρύπα ή απλά κυκλοειδής επιφάνεια χρησιμοποιώντας την πληροφορία του βάθους που μας παρέχει η στερεοσκοπική κάμερα.

Συνοπτικά τα βήματα της υλοποίησης μας είναι τα ακόλουθα:

- Είναι γνωστό ότι πιθανές θέσεις τρύπας είναι πάνω σε ξύλα. Συνεπώς μπορούμε να περιορίσουμε τις περιοχές ενδιαφέροντος της σκηνής μας σε εκείνες που βρίσκονται πάνω σε ξύλα και σε όλες τις υπόλοιπες. Η συγκεκριμένη παραδοχή είναι πολύ σημαντική καθώς περιορίζει σημαντικά το μέγεθος της περιοχής που ψάχνουμε κάθε φορά.

- Σε δεύτερο στάδιο εφαρμόζεται στην εικόνα μας κατάλληλο φίλτρο, το οποίο εντοπίζει όλες τις ακμές στις περιοχές που μας ενδιαφέρουν. Σε αυτό το σημείο και εφόσον έχουμε τις ακμές ψάχνουμε να βρούμε κυκλικές περιοχές που εμφανίζονται στην παρούσα εικόνα.

- Μέχρι αυτό το σημείο, δεν έχουμε καμία πληροφορία για το αν η κυκλική περιοχή που εντοπίσαμε στα προηγούμενα βήματα είναι όντως τρύπα. Για να μπορέσουμε επομένως να διαχωρίσουμε τις τρύπες από τα πιθανά false alarms χρησιμοποιούμε σε αυτό το σημείο την πληροφορία του βάθους που έχουμε εξασφαλίσει από την στερεοσκοπική κάμερα. Έτσι για κάθε μια από τις περιοχές ενδιαφέροντος (κυκλοειδείς επιφάνειες-blobs) συγκρίνουμε βάθη και ανάλογα καταλήγουμε στο αν είναι τρύπα ή όχι.

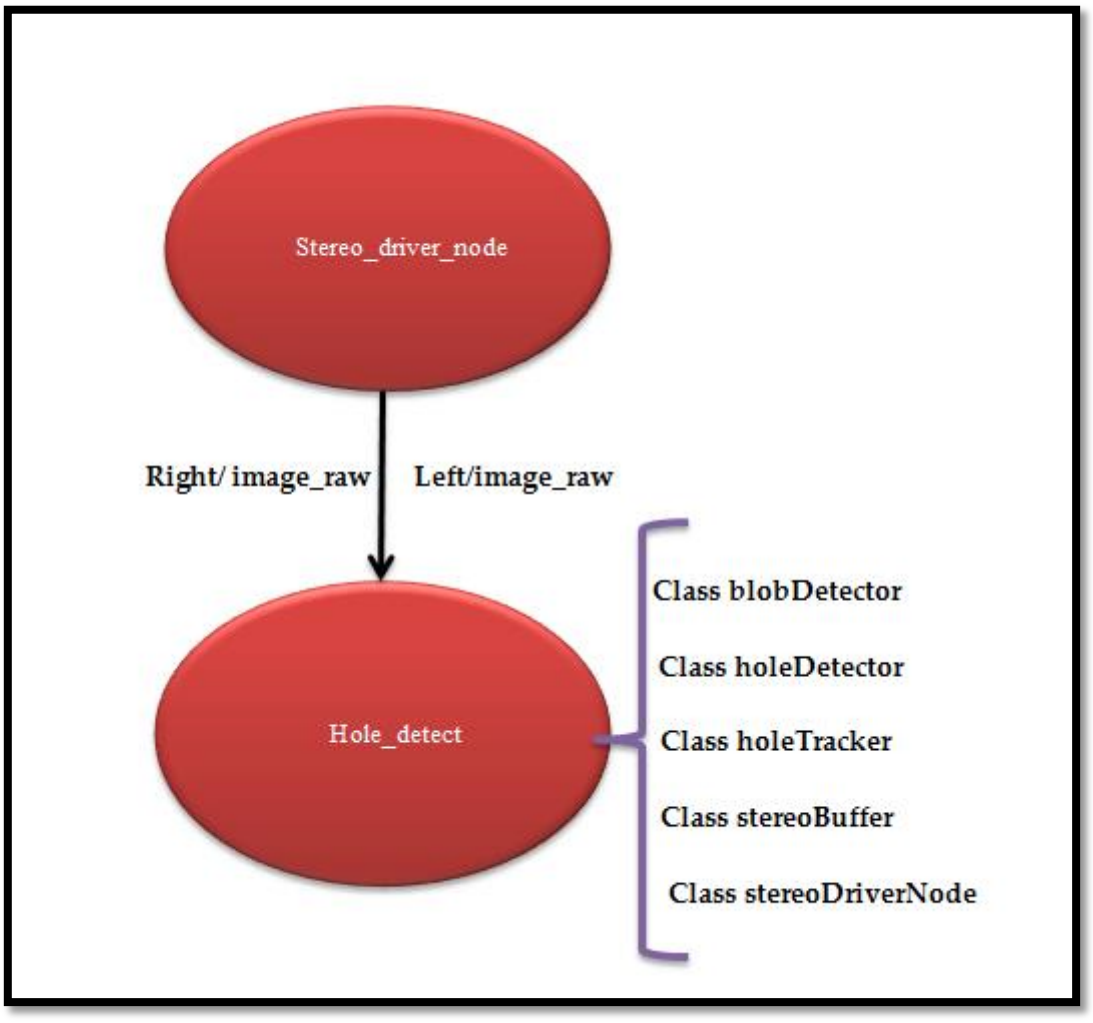
- Στην πορεία της υλοποίησης μας παρατηρήσαμε ότι η πληροφορία του βάθους που εξασφαλίζουμε σε μερικές περιπτώσεις έχει μεγάλα σφάλματα επομένως αποφασίσαμε να εισάγουμε μια επιπλέον δικλίδα ασφαλείας για να μειώσουμε την πιθανότητα σφάλματος. Έτσι για να καταλήξουμε ότι μια τρύπα είναι όντως τρύπα και όχι false alarm είναι αναγκαίο να εντοπιστεί σε 5 συνεχόμενα frames.

- Τέλος για να γνωρίζουμε ανά πάσα στιγμή τη θέση της τρύπας, που έχουμε εντοπίσει ,την κάνουμε track όσο αυτή εμφανίζεται στις εικόνες

που λαμβάνουμε από τις κάμερες.

2.2. Αναλυτική παρουσίαση του κώδικα για την εύρεση τρύπας με τη χρήση στερεοσκοπικής κάμερας

Σε αυτό το σημείο να προσθέσω ότι σε αντιστοιχία με το vision έτσι και εδώ δημιουργήσαμε έναν κόμβο, ο οποίος είναι υπεύθυνος για να ποστάρει εικόνες σε ένα topic και όλοι οι υπόλοιποι να ακούνε τα μηνύματα που ο τελευταίος αποστέλλει. Ο συγκεκριμένος κόμβος είναι ο stereoDriverNode ο οποίος εκτός από το προηγούμενο επιπρόσθετα συγχρονίζει τα μηνύματα που λαμβάνουμε από τις 2 κάμερες.



2.2.1. Class BlobDetector:

Η βασική συνάρτηση της συγκεκριμένης κλάσης είναι η findBlobs(). Όπως αναφέρθηκε παραπάνω στόχος της παρούσας κλάσης είναι η εύρεση blobs. Ο υπολογισμός των blobs βασίζεται στην εικόνα που παίρνουμε από την αριστερή κάμερα (left frame) και στον disparity map που προκύπτει.

Σε αυτό το σημείο καλό θα ήταν να παραθέσω ορισμένα στοιχεία για τον disparity map. **Disparity** σημαίνει διαφορά, έτσι θα μπορούσαμε απλά να πούμε ότι disparity map είναι ένα μέτρο της διαφοράς μεταξύ δυο πραγμάτων. Στην δικιά μας περίπτωση ως disparity map ορίζεται το μέτρο της διαφοράς μεταξύ των οπτικών γωνιών των δυο καμερών (left view and right view). Ας πούμε για παράδειγμα ότι από την αριστερή κάμερα εντοπίζουμε ένα pixel στην σκηνή μας, έχουμε τη δυνατότητα να εντοπίσουμε αν όχι ακριβώς το ίδιο pixel τότε σίγουρα κάποιο γειτονικό του στην εικόνα που βλέπουμε από την αριστερή κάμερα και να μετρήσουμε στην διαφορά μεταξύ της τετμημένης των δυο σημείων. Αν εφαρμόσουμε το παραπάνω για κάθε pixel παίρνουμε τον disparity map της σκηνής.

Οι δυο διαθέσιμοι αλγόριθμοι που είναι υλοποιημένοι στην openCV για τον υπολογισμό του disparity είναι ο StereoSGBM (semi block matching algorithm) και ο StereoBM (block matching algorithm). Στη συνέχεια παρατίθενται συνοπτικά ορισμένα χαρακτηριστικά των παραπάνω αλγορίθμων.

1. StereoBM:

Ο block matching algorithm υλοποιείται μέσω της κλάσης StereoBM, η οποία ορίζει αντικείμενα με τα ακόλουθα χαρακτηριστικά. Έπειτα από ποικιλία πειραμάτων καταλήξαμε στις ακόλουθες τιμές, τις οποίες και παραθέτω.

```
StereoBM sbm;
sbm.state->SADWindowSize = 9;
sbm.state->numberOfDisparities = 112;
sbm.state->preFilterSize = 5;
sbm.state->preFilterCap = 61;
sbm.state->minDisparity = -39;
sbm.state->textureThreshold = 507;
sbm.state->uniquenessRatio = 0;
sbm.state->speckleWindowSize = 0;
sbm.state->speckleRange = 8;
sbm.state->disp12MaxDiff = 1;
```

Αναλυτικά οι παραπάνω τιμές είναι:

- minDisparity – Minimum possible disparity value.
- numDisparities – Maximum disparity minus minimum disparity. This parameter must be divisible by 16.
- SADWindowSize – Matched block size. It must be an odd number >=1
- disp12MaxDiff – Maximum allowed difference (in integer pixel units) in the left-right disparity check.
- preFilterCap – Truncation value for the prefiltered image pixels.

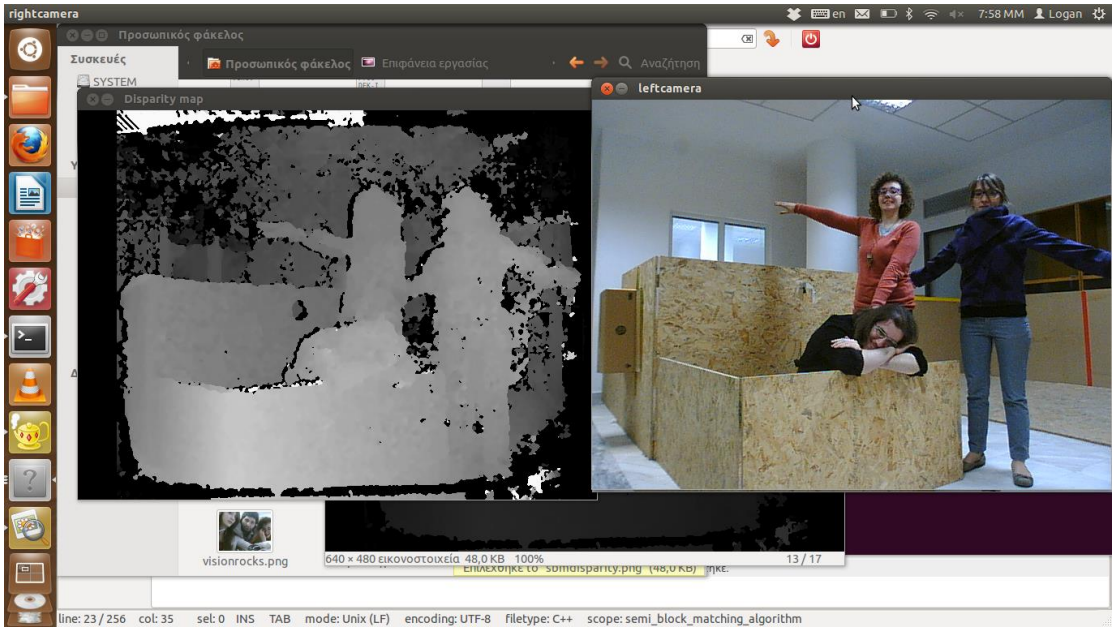
- uniquenessRatio – Margin in percentage by which the best (minimum) computed cost function value should “win” the second best value to consider the found match correct. Normally, a value within the 5-15 range is good enough.
- speckleWindowSize – Maximum size of smooth disparity regions to consider their noise speckles and invalidate.
- speckleRange – Maximum disparity variation within each connected component.

2. StereoSGBM:

Κατά αντιστοιχία ο semi block matching algorithm υλοποιείται μέσω της κλάσης StereoSGBM, η οποία ορίζει αντικείμενα με τα ακόλουθα χαρακτηριστικά. Έπειτα από ποικιλία πειραμάτων καταλήξαμε στις ακόλουθες τιμές, τις οποίες και παραθέτω.

```
StereoSGBM sgbm;
sgbm.SADWindowSize = 5;
sgbm.numberOfDisparities = 192;
sgbm.preFilterCap = 4;
sgbm.minDisparity = -64;
sgbm.uniquenessRatio = 1;
sgbm.speckleWindowSize = 150;
sgbm.speckleRange = 2;
sgbm.disp12MaxDiff = 10;
sgbm.fullDP = false;
sgbm.P1 = 600;
sgbm.P2 = 2400;
```

Έπειτα από διάφορα πειράματα που εκτελέσαμε καταλήξαμε ότι για τις δικές μας απαιτήσεις ο semi block matching αλγόριθμος παρείχε πιο ικανοποιητικά αποτελέσματα. Παραθέτω ενδεικτικά μια εικόνα πριν και μετά την εφαρμογή του semi block matching αλγόριθμου.



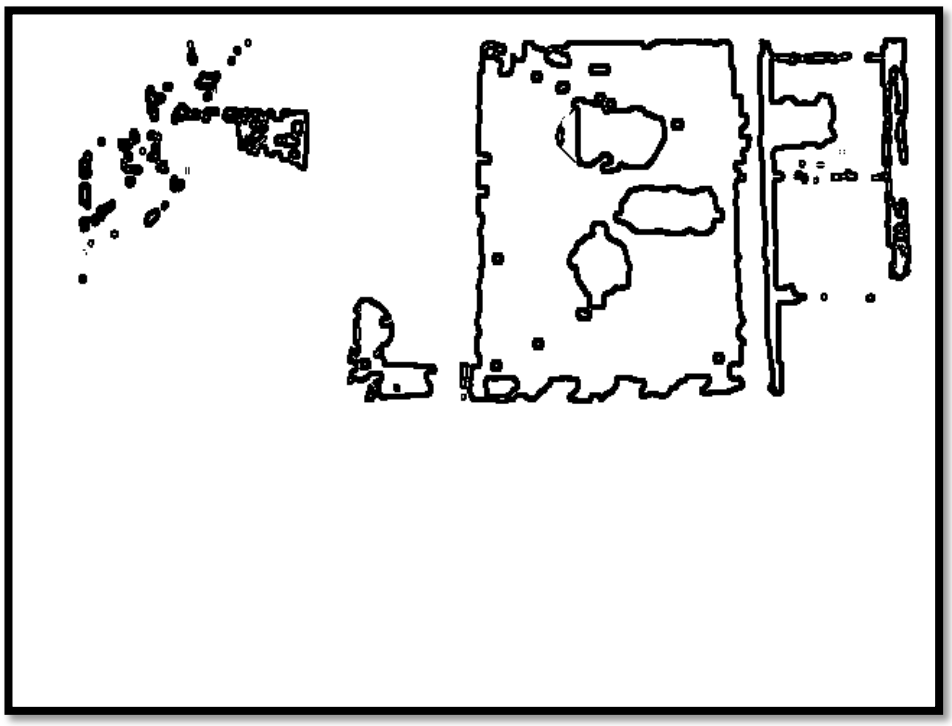
Σε αυτό το σημείο να σημειώσω ότι εναλλακτικά ο υπολογισμός του disparity map είναι εφικτός μέσω έτοιμων πακέτων του Ros. Στην υλοποίηση μας επιλέξαμε να χρησιμοποιήσουμε τον disparity, που προκύπτει από έτοιμα πακέτα του Ros καθώς είδαμε ότι έδινε καλύτερα αποτελέσματα σε διάφορα πειράματα που εκτελέσαμε.

Όπως ανέφερα και παραπάνω ένα κρίσιμο βήμα στον περιορισμό των περιοχών που πρέπει να εξετάσουμε για παρουσία τρύπας είναι να διαχωρίσουμε τις περιοχές τις εικόνες που έχουν ξύλο από εκείνες που δεν έχουν. Η συγκεκριμένη παραδοχή είναι αρκετά ασφαλής δεδομένου ότι γνωρίζουμε με βεβαιότητα ότι οι μοναδικές πιθανές θέσεις τρύπας είναι πάνω σε ξύλα.

Συνεπώς σε πρώτο βήμα υπολογίζουμε ένα ιστόγραμμα, με βάση διάφορες εικόνες με ξύλα της πίστας που έχουμε τραβήξει προηγουμένως. Ο λόγος που φτιάχνουμε το συγκεκριμένο ιστόγραμμα είναι για μπορέσουμε να εντοπίσουμε τα χαρακτηριστικά, βάσει των οποίων το τελευταίο δημιουργήθηκε στην εικόνα μας. Πιο αναλυτικά επιθυμία μας είναι να δούμε σε ποιες περιοχές τις εικόνας υπάρχει ξύλο για να μπορέσουμε να αποκλείσουμε όλες τις υπόλοιπες. Έτσι για δεδομένο ξύλο βρίσκουμε ένα histogram model. Σε επόμενο βήμα θέλουμε να δούμε πόσα καλά τα pixels της εικόνας ταιριάζουν στην κατανομή του histogram model που δημιουργήσαμε προηγουμένως. Ο τρόπος για να γίνει αυτό είναι να υπολογιστεί το backprojection.

Σε επόμενη φάση συνδυάζοντας την πληροφορία από τον backprojection και τον disparity δημιουργείται μια νέα εικόνα, στην οποία περιέχεται πληροφορία μόνο για τις περιοχές που υπάρχει ξύλο. Σε αυτό το στάδιο εκτελούμε αναζήτηση ακμών στην νέα εικόνα που έχει προκύψει. Δυο από τους διαθέσιμους αλγορίθμους για edge detection, υλοποιημένους στην openCV είναι ο Sobel και ο Canny. Έπειτα από πειραματισμούς και σύγκριση των αποτελεσμάτων που

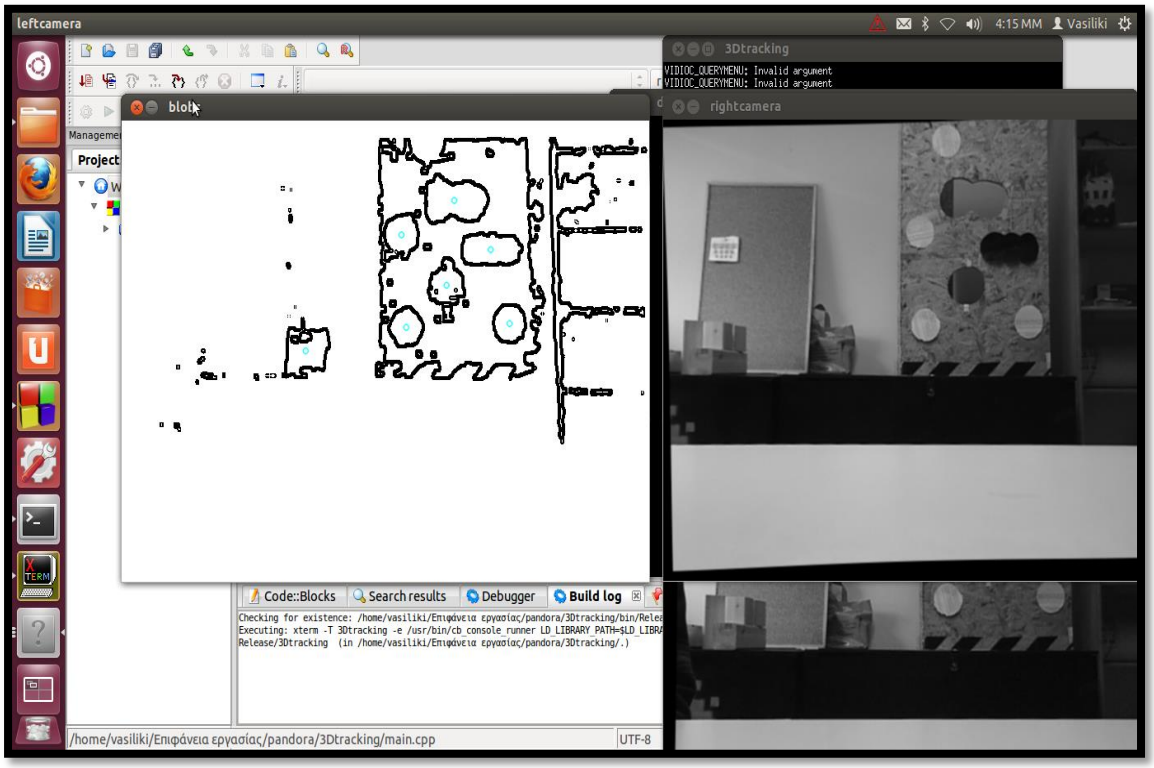
προκύπτουν από τους δυο αλγορίθμους επιλέξαμε τον Sobel. Ενδεικτικά παραθέτω μια εικόνα που προέκυψε από την εφαρμογή του συγκεκριμένου αλγορίθμου.



Τέλος στη νέα εικόνα που έχουμε επεξεργαστεί κατάλληλα σύμφωνα με όλα τα προηγούμενα εφαρμόζουμε blobDetection για να βρούμε τα διάφορα blobs της εικόνας. Στην openCV υλοποιείται για τον συγκεκριμένο σκοπό η κλάση SimpleBlobDetector, η οποία εντοπίζει τα blobs σε μια εικόνα. Για καλύτερη κατανόηση όλων των παραπάνω παραθέτω την ακόλουθη εικόνα, στην οποία σημειώνονται όλες εκείνες οι περιοχές, τις οποίες ο αλγόριθμος έχει αναγνωρίσει ως blobs. Σε αυτό το σημείο να προσθέσω ότι οι θέσεις των σημείων στην εικόνα που έχουν μπλε χρώμα, αναφέρονται στις συντεταγμένες των κέντρων των blobs πάνω στην εικόνα. Σε αυτό το σημείο θα ήθελα να σημειώσω ότι κάθε αντικείμενο της κλάσης SimpleBlobDetector έχει ορισμένα χαρακτηριστικά, βάσει των οποίων αναγνωρίζονται τα blobs σε μια εικόνα. Ενδεικτικά παραθέτω τις τιμές στις οποίες εμείς καταλήξαμε έπειτα από πειραματισμούς.

```
cv::SimpleBlobDetector::Params params;
 params.minThreshold = 40;
 params.maxThreshold = 60;
 params.thresholdStep = 5;
 params.minArea = 550;
 params.minConvexity = 0.6;
 params.minInertiaRatio = 0.5;
 params.maxArea = 8000;
 params.maxConvexity = 10;
 params.maxCircularity=1;
 params.minCircularity=0.3;
 params.filterByColor = false;
```

params.filterByCircularity = true;



2.2.2. Class HoleDetector:

Μέχρι στιγμής βάσει της υλοποίησης μας είμαστε σε θέση να αναγνωρίσουμε κυκλοειδείς επιφάνειες σε μια εικόνα. Δεν έχουμε ωστόσο καμία πληροφορία για το ποιες από αυτές είναι όντως τρύπες (έχουν βάθος) ή απλά είναι κάποιο κολλημένο χαρτί με κυκλικό σχήμα. Παρατηρώντας την παραπάνω εικόνα βλέπουμε εύκολα ακριβώς αυτό, ότι δηλαδή μέχρι στιγμής το μόνο που έχουμε καταφέρει είναι να εντοπίζουμε περιοχές που προσεγγίζουν σχηματικά κύκλους. Συνεπώς η βασική λειτουργική απαίτηση που καλείται να καλύψει η συγκεκριμένη κλάση είναι να διαχωρίσει τις τρύπες από τα όλες τις υπόλοιπες κυκλικές περιοχές λαμβάνοντας υπόψη διαφορές στο βάθος.

Η βασική συνάρτηση της συγκεκριμένης κλάσης είναι η defineHoles() . Αρχικά μέσω ενός αντικειμένου της κλάσης BlobDetector καλείται η συνάρτηση findBlobs() για τον εντοπισμό των blobs στην εικόνα που βλέπουν οι κάμερες. Σε επόμενο βήμα καλώντας την συνάρτηση reprojectImageTo3D() της openCV υπολογίζεται ο **point cloud** όλων των σημείων της εικόνας. Ως point cloud ορίζεται ένα σύνολο από σημεία σε ένα σύστημα συντεταγμένων. Στην δικιά μας περίπτωση ως point cloud ορίζονται οι τρισορθογώνιες συντεταγμένες (X,Y,Z) όλων των σημείων στο χώρο. Αρχή Ο (0,0) του συστήματος συντεταγμένων μας θεωρείται το κέντρο του φακού της αριστερής κάμερας. Συνεπώς όλοι οι υπολογισμοί πραγματοποιούνται θεωρώντας ως αρχή το προαναφερθέν σημείο.

Σε αυτό το σημείο με βάση τις θέσεις των blobs πάνω στην εικόνα που έχει υπολογίσει το αντικείμενο της κλάσης blobDetector δημιουργούνται δύο καινούργιες εικόνες, μια που περιέχει μόνο τα εντοπισμένα blobs γεμισμένα και μια δεύτερη που περιέχει τα ίδια blobs, αφού έχει αυξηθεί ο κυκλικός δίσκος που κάθε ένα από αυτά ορίζει . Στη συνέχεια υπολογίζουμε την μέση τιμή του point cloud στις δυο προηγούμενες εικόνες σε εκείνες τις περιοχές που έχουν οριστεί τα blobs. Συγκρίνοντας ακόλουθα τις τιμές αυτές προκύπτει μια πιθανότητα βάσει της οποίας καταλήγουμε αν είναι τρύπα ή όχι.

Όπως ανέφερα και παραπάνω στην πορεία της υλοποίησης μας, παρατηρήσαμε ότι οι τιμές που μας έδινε ο point cloud συχνά είχαν μεγάλα σφάλματα. Επομένως για να εισάγουμε μια δικλείδα ασφαλείας δημιουργήσαμε έναν buffer στον οποίο κρατούσαμε πληροφορία για 5 συνεχόμενα frames. Σε περίπτωση που κάποια τρύπα δεν εμφανιστεί τουλάχιστον σε 5 συνεχόμενα frames, θεωρούμε ότι δεν είναι τρύπα και την απορρίπτουμε. Από την άλλη έστω ότι ένα blob εντοπίζεται σε 5 συνεχόμενες εικόνες που παίρνουμε από την κάμερα, σε εκείνη την περίπτωση υπολογίζεται ο μέσος όρος των μέσων όρων που έχουν υπολογιστεί και για τα 5 προηγούμενα frames και αναλόγως με τη διαφορά βάθους διαμορφώνεται η πιθανότητα. Οι τιμές για τα threshold των πιθανοτήτων, στις οποίες καταλήξαμε προέκυψαν έπειτα από πειραματισμό και ενδεχόμενα επιδέχονται βελτίωση.

Σε αυτό το σημείο απλά να προσθέσω ότι ο buffer που προανέφερα δεν υλοποιείται από κάποια έτοιμη δομή της c++, αλλά είναι αντικείμενο της κλάσης stereoBuffer. Συνοπτικά να σημειώσω ότι αντικείμενα της συγκεκριμένης κλάσης δημιουργούνται και καταστρέφονται ανά 5 frames και περιέχουν πληροφορία για τα ακόλουθα:

- Disparity map, που έχει προκύψει από κάθε εικόνα που παίρνουμε από τις κάμερες, όπως αυτό υπολογίζεται μέσω ros.
- LeftFrame, η εικόνα από την αριστερή κάμερα.
- KeyPoints, οι συντεταγμένες πάνω στην εικόνα των κέντρων των blobs που έχουν εντοπιστεί.
- Probability, οι διάφορες πιθανότητες που προκύπτουν για κάθε blob σε κάθε frame.

2.2.3. Class HoleTracker:

Όπως πολύ εύκολα γίνεται αντιληπτό από όλα τα παραπάνω στόχος του συγκεκριμένου κόμβου είναι ο εντοπισμός τρύπας σύμφωνα με πληροφορία που λαμβάνουμε από την στερεοσκοπική κάμερα. Επομένως το συγκεκριμένο node

πρέπει αφενός προφανώς να εντοπίζει τρύπες και αφετέρου να ενημερώνει το data fusion για την ακριβή θέση της τρύπας και για την πιθανότητα να είναι τρύπα. Στην προηγούμενη κλάση καταλήγαμε αν ένα blob είναι τρύπα ή false alarm από την πληροφορία που εξασφαλίζαμε από 5 συνεχόμενα frames, στο οποίο το τελευταίο εμφανιζόταν. Εδώ ανάλογα με το πλήθος των frames, στα οποία μια τρύπα εμφανίζεται διαμορφώνεται η πιθανότητα που αποστέλλεται τελικά στο data fusion.

Για να υλοποιηθεί το παραπάνω είναι απαραίτητο να αναπτυχθεί μια μέθοδος tracking. Συγκεκριμένα εμείς υλοποιήσαμε τον Camshift αλγόριθμο για να μπορέσουμε να κάνουμε track την τρύπα.