

Cairo University

Faculty of Engineering

Department of Computer Engineering

**Project Documentation**

**Compiler C**

Report to explain the approach we took to make a tracking module using Mosse Tracker

**Introduction**

In this document we present the approach we made to make a compiler for language similar to c/c++.

We list the five big modules we had to make:-

[1] lexer

[2] parser

[3] quadrable “assembly” language generation

[4] semantic analysis

[5] syntax error handler

**Compiler**

We implemented in our compiler:-

|  |  |
| --- | --- |
| Variables and Constants declaration | int x; |
| const int x; |
| float y; |
| const float y; |
| Mathematical expressions | x + y |
| x – y |
| x \* y |
| -x |
| x ++ |
| x -- |
| logical expressions | x == y |
| x != y |
| x > y |
| x < y |
| x >= y |
| x <= y |
| Assignment statement | x = y;  x = “mathematical expressions” |
| If then else | if ( “logical expression” ) { “stmts” } |
| else if ( “logical expression” ) { “stmts” } |
| else { “stmts”} |
| while | while ( “logical expression” ) { “stmts” } |
| Do while | do { “stmts” } while ( “logical expression” ) |
| For loop | For ( “Assignment statement” ; “logical expression” ; “Assignment statement” ) { “stmts” } |
| Block Structure | Global |
| Function block |
| If block |
| For block |
| Do-while block |
| While block |
| Functions | int add(int n1, int n2){ “stmts”} |
| Function Call | x = add( 1,2); |
| y = add(1.1,2); |
| Function Overloading | int add(int n1, int n2){ “stmts”} |
| float add(float n1, int n2) |

**LEXER**

|  |  |
| --- | --- |
| User code | Lexer output |
| { | { |
| } | } |
| [ | [ |
| ] | ] |
| ( | ( |
| ) | ) |
| ; | ; |
| . | . |
| , | , |
| = | = |
| - | MINUS |
| + | PLUS |
| \* | MUL |
| / | DIV |
| <= | LE |
| >= | GE |
| < | LT |
| > | GT |
| == | EQ |
| != | NE |
| ++ | PP |
| -- | MM |
| int | INT |
| float | FLOAT |
| const | CONST |
| if | IF |
| else | ELSE |
| do | DO |
| while | WHILE |
| for | FOR |
| return | RETURN |
| [0-9]+ | INTNUM |
| [0-9]+.[0-9]+ | FLOATNUM |
| [A-Za-z][A-Za-z0-9\_]\* | ID |
| [ \t\c] | - |
| "\n" | - |

**PARSER**

We splited our grammer to three main category

* Program => Globla Variable Declaration and all the functions and types
* Statements => every statement in the program.
* Expressions => every expression mentioned above + function calls.

|  |  |  |
| --- | --- | --- |
| Program | Program | Declaration\_List Function\_List |
| Declaration\_List |
| Function\_List |
| Declaration\_List | Type ID ';' |
| Declaration\_List Type ID ';' |
| Function\_List | Type ID '(' ')' Stmt\_Group |
| Function\_List Type ID '(' Parameter\_List ')' Stmt\_Group |
| Type ID '(' Parameter\_List ')' Stmt\_Group |
| Function\_List Type ID '(' ')' Stmt\_Group |
| Parameter\_List | Type ID |
| Parameter\_List ',' Type ID |
| Arg\_List | Expr |
| Arg\_List ',' Expr |
| Type | INT |
| FLOAT |
| CONST INT |
| CONST FLOAT |
| Statements | Stmt | ID '=' Expr ';' |
| ID '(' ')' ';' |
| ID '(' Arg\_List ')' ';' |
| RETURN ';' |
| RETURN Expr ';' |
| IF '(' Expr ')' Stmt %prec NO\_ELSE |
| IF '(' Expr ')' Stmt ELSE Stmt |
| FOR '(' ID '=' Expr ';' Expr ';' ID '=' Expr ')' Stmt |
| WHILE '(' Expr ')' Stmt |
| DO Stmt WHILE '(' Expr ')' ';' |
| '{' Declaration\_List Stmt\_List '}' |
| '{' Declaration\_List '}' |
| '{' Stmt\_List '}' |
| '{' '}' |
| ID PP |
| ID MM |
| ';' |
| Stmt\_Group | '{' Declaration\_List Stmt\_List '}' |
| '{' Declaration\_List '}' |
| '{' Stmt\_List '}' |
| '{' '}' |
| Stmt\_List | Stmt |
| Stmt\_List Stmt |
| Expressions | Expr | Expr MINUS Expr |
| Expr PLUS Expr |
| Expr MUL Expr |
| Expr DIV Expr |
| MINUS Expr %prec UNARY |
| Expr LE Expr |
| Expr GE Expr |
| Expr GT Expr |
| Expr LT Expr |
| Expr EQ Expr |
| Expr NE Expr |
| '(' Expr ')' |
| ID '(' ')' |
| ID '(' Arg\_List ')' |
| INTNUM |
| FLOATNUM |
| ID |

**Implementation**

Here are the steps that we implemented in the program for Tracking Module “Mosse Tracker”:-

Input: gray\_frame,dimensions of the car in the frame, frame\_width, frame height and tracker\_id: Sequence of frames.

Output : tacked car.

1. Extract the car image from whole frame.
2. Make a window and set its center to one.
3. Run gaussianblur on it with segma = 3 //according to paper segma = 2 but we found this is better in our problem.
4. Normalize the window to become [0,1].
5. Get the fast fourier transform to the window.
6. Make another two windows of the same size of first and assign all its values to zero as we will use it later to calculate the numenator and denominator in fig(1).
7. Make gaussian blur to cut image //not mentioned in paper but found it get better results in noisy images.
8. Start loop on num\_of\_training\_imgs //advised in the paper to make 125 training numbers but found that is bad for our problem as our problem doesn’t have lots of rotations so tried different numbers and found best is 25.
9. Make random rotations.
10. Make preprocess to image as paper stated and illustrated above.
11. Then get fourier transform to it.
12. Multiply G with F\* and F with F\*

fig(1)

1. Then train all the num\_of\_training images and update the filter and calulate H\*.
2. Now we start to update the tracking this is the step we will use each time we want to update tracking.
3. Extract the car frame , then make gaussian blur then make preprocessing stage then start to correlate this image with the filter.
4. Convert car image into fourier transform F.
5. Multiply F with H\*

fig(2)

1. Convert the output G to spatial domain using inverse fast fourier transform as we stated that above.
2. Calculate the max value in the matrix and get its position // the max value means this is the highest correlation point between the new image and the filter.
3. The peak position is the new position for the tracked car.
4. Then calculate the psr according to the paper , get the mean and standard deviation.
5. After we calculate the psr we compare its result if it’s lower than 7 then the tracker has lost the car.
6. If it’s higher, then make an online training update to the new image using what we illustrated above from step 12 to 17 and update the old values with new by learning rate of 0.225 .
7. And so on every new frame goes from step 14 to 23 for 30 frames.
8. Then save all these 30 frames to disk or enter it directly to the vif descriptor we will make later.

**References**

[1] [D. S. Bolme, J. R. Beveridge, B. A. Draper and Y. M. Lui, "Visual object tracking using adaptive correlation filters," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp. 2544-2550.](https://ieeexplore.ieee.org/document/5539960)