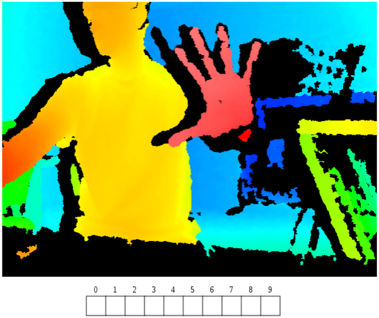
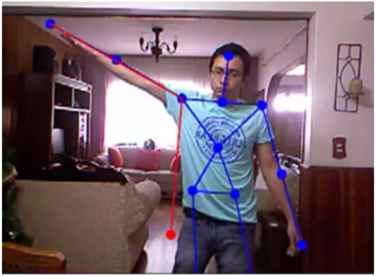
K-Da Library

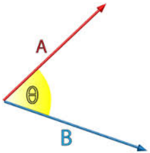
Review

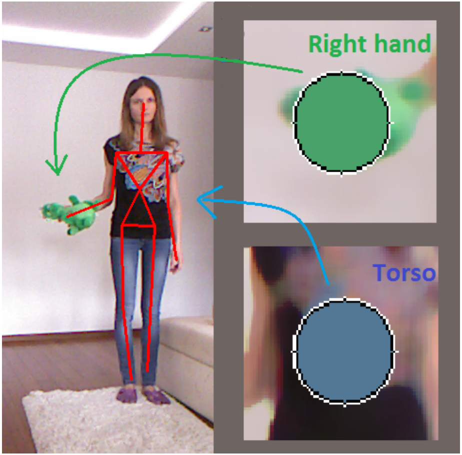




Class conversion

- convertir(string pjoint1, string pjoint2, int n);
- llenarArregloAngulos();
- getArregloAngulos();





$$\vec{A}\cdot\vec{B} = A\,B\cos\theta$$

Class compara

- sacapromedios(double arreglo);
- arreglo_promedio(double arreglo_prom1, double arreglo_prom2);

sacapromedios

Array recibido:

$[n, k, ..., l, m]$

Array retornado:

$[prom(x1, x2, ..., x10)]$

arreglo_promedio

Array recibido:

$[prom1(x1, x2, ..., x10)]$

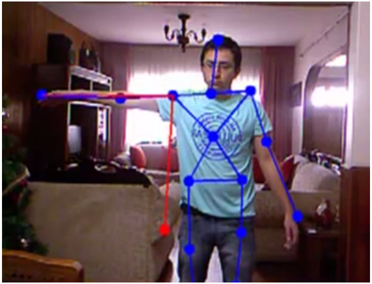
$[prom2(x1, x2, ..., x10)]$

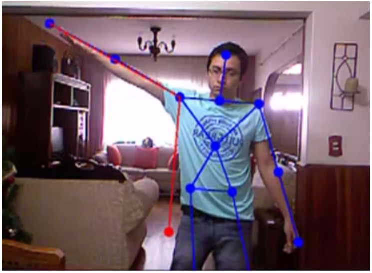
Array retornado:

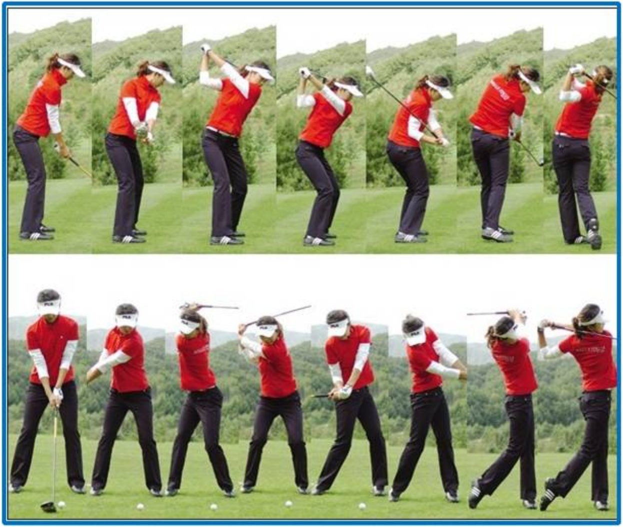
$[1, 0, 0, 1, 0, 1, 1, 0, 1, 1]$

Class compara

- comparar_angulos(int promedio);
- comparar_velocidad(int pSizeMov1, int pSizeMov2);





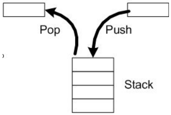


Problems of the previous presentation

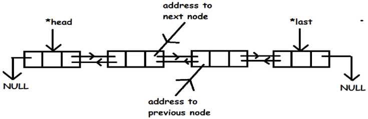
- Kinect-data
- Speed analysis

Data structures to implement

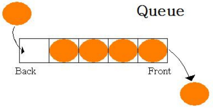
- Stack



- Double linked list



- Queue



Complexity Analysis

Data Estructure	Storage	Using the methods
Stack	O(1)	O(n)
Linked list	O(1)	O(n)
Queue	O(1)	O(n)

Complexity analysis in our comparison algorithm

A=[1,2,....,n]

for (i=1; $i <= n$; i++)

linea i = A[i];

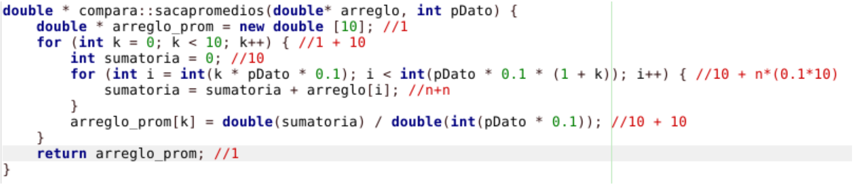
1

1 + n

3n

4n + 2

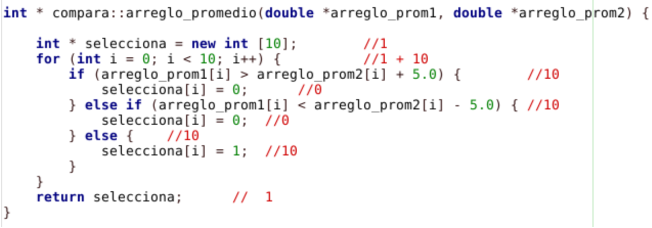
Complexity analysis in our comparison algorithm



```
double * compara::sacapromedios(double* arreglo, int pDato) {
    double * arreglo_prom = new double [10]; //1
    for (int k = 0; k < 10; k++) { //1 + 10
        int sumatoria = 0; //10
        for (int i = int(k * pDato * 0.1); i < int(pDato * 0.1 * (1 + k)); i++) { //10 + n*(0.1*10)
            sumatoria = sumatoria + arreglo[i]; //n+n
        }
        arreglo_prom[k] = double(sumatoria) / double(int(pDato * 0.1)); //10 + 10
    }
    return arreglo_prom; //1
}
```

$3n + 53$

Complexity analysis in our comparison algorithm



```
int * compara::arreglo_promedio(double *arreglo_prom1, double *arreglo_prom2) {

    int * selecciona = new int [10];        //1
    for (int i = 0; i < 10; i++) {          //1 + 10
        if (arreglo_prom1[i] > arreglo_prom2[i] + 5.0) {        //10
            selecciona[i] = 0;      //0
        } else if (arreglo_prom1[i] < arreglo_prom2[i] - 5.0) { //10
            selecciona[i] = 0;  //0
        } else {    //10
            selecciona[i] = 1;  //10
        }
    }
    return selecciona;      //  1
}
```

53
Total 3n + 106
Orden de Complejidad es de O(n)