En el siguiente datasheet se expondrán de forma ordenada y detallada, las principales y más importantes funciones y tipos de datos utilizados en la realización de la aplicación.

DATOS:

IplImage - Inicializa un dato tipo IplImage para almacenar las imágenes.

CvCapture - Inicializa un dato tipo CvCapture (vídeo en formato de OpenCV preparado para ser procesado frame a frame).

CBlobResult – Inicializa un dato tipo CBlobResult que se utiliza para manipular los blobs que aparecen.

CBlob – Inicializa un dato tipo CBlob para almacenar en cada momento el blob sobre el que se quiere trabajar.

CvPoint – Inicializa un dato tipo CvPoint que sirve para guardar las coordenadas del punto que se desee.

CvFont – Inicializa un dato tipo CvFont para escribir texto sobre la ventana que se desee.

CvBGStatModel - Inicializa un dato tipo CvBGStatModel para crear el modelo.

FGDStatModelParams – Inicializa una estructura FGDStatModelParams donde se definen los parámetros del modelo.

```
typedef struct CvFGDStatModelParams
{
        int    Lc;
        int    N1c;
        int    N2c;

        int    Lcc;
        int    N1cc;
        int    N2cc;

        int    is_obj_without_holes;
        int    perform_morphing;

        float  alpha1;
        float  alpha2;
        float  alpha3;

        float  delta;
        float  T;
        float  minArea;
}
```

IplConvKernel – Inicializa un dato tipo IplConvKernel que se utiliza para realizar un filtro morfológico.

CvKalman – Inicializa un dato tipo CvKalman que se utiliza para realizar un filtro de Kalman.

```
typedef struct CvKalman

{

        int MP;
        int DP;
        int CP;

        float* PosterState;
        float* PriorState;
        float* DynamMatr;
        float* MeasurementMatr;
        float* MNCovariance;
        float* PNCovariance;
        float* KalmGainMatr;
        float* PriorErrorCovariance;
        float* PosterErrorCovariance;
        float* Temp1;
        float* Temp2;

        CvMat* state_pre;
        CvMat* state_post;
        CvMat* transition_matrix;
        CvMat* control_matrix;
        CvMat* measurement_matrix;
        CvMat* process_noise_cov;
        CvMat* measurement_noise_cov;
        CvMat* error_cov_pre;                                                   /
        CvMat* gain;
        CvMat* error_cov_post;

        CvMat* temp1;
        CvMat* temp2;
        CvMat* temp3;
        CvMat* temp4;
        CvMat* temp5;

}

CvKalman;
```

FUNCIONES:

```
/* Just a combination of cvGrabFrame and cvRetrieveFrame
   !!!DO NOT RELEASE or MODIFY the retrieved frame!!!      */
CVAPI(IplImage*) cvQueryFrame( CvCapture* capture );

/* stop capturing/reading and free resources */
CVAPI(void) cvReleaseCapture( CvCapture** capture );

/* Creates FGD model */
CVAPI(CvBGStatModel*) cvCreateFGDStatModel( IplImage* first_frame, CvFGDStatModelParams*
parameters CV_DEFAULT(NULL));

/*Releases memory used by BGStatModel*/
CV_INLINE void cvReleaseBGStatModel( CvBGStatModel** bg_model )
{
    if( bg_model && *bg_model && (*bg_model)->release )
        (*bg_model)->release( bg_model );

/* start capturing frames from camera: index = camera_index + domain_offset (CV_CAP_*) */

#define cvCaptureFromCAM cvCreateCameraCapture
CVAPI(CvCapture*) cvCreateCameraCapture( int index );


extern "C" IplConvKernel* cvCreateStructuringElementEx(
            int cols, int rows, int anchor_x, int anchor_y,
            int shape, int* values = NULL );


extern "C" void cvErode( const CvArr* src, CvArr* dst,
                    IplConvKernel* element = NULL,
                    int iterations = 1 );

extern "C" void cvDilate( const CvArr* src, CvArr* dst,
                     IplConvKernel* element = NULL,
                     int iterations = 1 );

class CBlobResult
{
public:

        //! constructor estandard, crea un conjunt buit de blobs
        //! Standard constructor, it creates an empty set of blobs
        CBlobResult();

        //! constructor a partir d'una imatge
        //! Image constructor, it creates an object with the blobs of the image
        CBlobResult(IplImage *source, IplImage *mask, int threshold, bool findmoments);
```

```
        //! constructor de còpia
        //! Copy constructor
        CBlobResult( const CBlobResult &source );

        //! Destructor
        virtual ~CBlobResult();


        //! operador = per a fer assignacions entre CBlobResult
        //! Assigment operator
        CBlobResult& operator=(const CBlobResult& source);

        //! operador + per concatenar dos CBlobResult
        //! Addition operator to concatenate two sets of blobs
        CBlobResult operator+( const CBlobResult& source );

        //! Afegeix un blob al conjunt
        //! Adds a blob to the set of blobs
        void AddBlob( CBlob *blob );


#ifdef MATRIXCV_ACTIU

        //! Calcula un valor sobre tots els blobs de la classe retornant una MatrixCV
        //! Computes some property on all the blobs of the class
        double_vector GetResult( funcio_calculBlob *evaluador ) const;

#endif

        //! Calcula un valor sobre tots els blobs de la classe retornant un std::vector<double>
        //! Computes some property on all the blobs of the class
        double_stl_vector GetSTLResult( funcio_calculBlob *evaluador ) const;

        //! Calcula un valor sobre un blob de la classe
        //! Computes some property on one blob of the class
        double GetNumber( int indexblob, funcio_calculBlob *evaluador ) const;

        //! Retorna aquells blobs que compleixen les condicions del filtre en el destination
        //! Filters the blobs of the class using some property

        void Filter(CBlobResult &dst,
                                int filterAction, funcio_calculBlob *evaluador,
                                int condition, double lowLimit, double highLimit = 0 );

        //! Retorna l'enèssim blob segons un determinat criteri
        //! Sorts the blobs of the class acording to some criteria and returns the n-th blob
        void GetNthBlob( funcio_calculBlob *criteri, int nBlob, CBlob &dst ) const;
```

```
        //! Retorna el blob enèssim
        //! Gets the n-th blob of the class ( without sorting )
        CBlob GetBlob(int indexblob) const;
        CBlob *GetBlob(int indexblob);

        //! Elimina tots els blobs de l'objecte
        //! Clears all the blobs of the class
        void ClearBlobs();


        //! Escriu els blobs a un fitxer
        //! Prints some features of all the blobs in a file
        void PrintBlobs( char *nom_fitxer ) const;

//Metodes GET/SET

        //! Retorna el total de blobs
        //! Gets the total number of blobs
        int GetNumBlobs() const
        {
                return(m_blobs.size());
        }

private:

        //! Funció per gestionar els errors
        //! Function to manage the errors

        void RaiseError(const int errorCode) const;

protected:

        //! Vector amb els blobs
        //! Vector with all the blobs
        blob_vector             m_blobs;
};


#ifdef BLOB_OBJECT_FACTORY

        /*Funció per comparar dos identificadors dins de la fàbrica de COperadorBlobs*/
        struct functorComparacioIdOperador
        {
          bool operator()(const char* s1, const char* s2) const
          {
                return strcmp(s1, s2) < 0;
          }
        };
```

```
//! Definition of Object factory type for COperadorBlob objects
typedef ObjectFactory<COperadorBlob, const char *, functorComparacioIdOperador >
t_OperadorBlobFactory;

//! Classe per calcular l'àrea d'un blob
//! Class to get the area of a blob

class CBlobGetArea : public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.Area();
        }
        const char *GetNom() const
        {
                return "CBlobGetArea";
        }
};


//! Classe per calcular la diferència en X del blob
class CBlobGetDiffX: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.maxx - blob.minx;
        }
        const char *GetNom() const
        {
                return "CBlobGetDiffX";
        }
};

//! Classe per calcular la diferència en Y del blob
class CBlobGetDiffY: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.maxy - blob.miny;
        }
        const char *GetNom() const
        {
                return "CBlobGetDiffY";
        }
};
```

```
//! Classe per a calcular la x mínima
//! Class to get the minimum x
class CBlobGetMinX: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.MinX();
        }
        const char *GetNom() const
        {
                return "CBlobGetMinX";
        }
};


//! Classe per a calcular la x màxima
//! Class to get the maximum x
class CBlobGetMaxX: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.MaxX();
        }
        const char *GetNom() const
        {
                return "CBlobGetMaxX";
        }
};

//! Classe per a calcular la y mínima
//! Class to get the minimum y
class CBlobGetMinY: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.MinY();
        }
        const char *GetNom() const
        {
                return "CBlobGetMinY";
        }
};


//! Classe per a calcular la y màxima
```

```
//! Class to get the maximum y
class CBlobGetMaxY: public COperadorBlob
{
public:
   double operator()(const CBlob &blob) const
        {
                return blob.MaxY();
        }
        const char *GetNom() const
        {
                return "CBlobGetMax";
        }
};
```

```
/* Creates Kalman filter and sets A, B, Q, R and state to some initial values */

CVAPI(CvKalman*) cvCreateKalman( int dynam_params, int measure_params,

                        int control_params CV_DEFAULT(0));



/* Releases Kalman filter state */

CVAPI(void)  cvReleaseKalman( CvKalman** kalman);



/* Updates Kalman filter by time (predicts future state of the system) */

CVAPI(const CvMat*)  cvKalmanPredict( CvKalman* kalman,

                              const CvMat* control CV_DEFAULT(NULL));



/* Updates Kalman filter by measurement
 (corrects state of the system and internal matrices) */

CVAPI(const CvMat*)  cvKalmanCorrect( CvKalman* kalman, const CvMat* measurement );
```