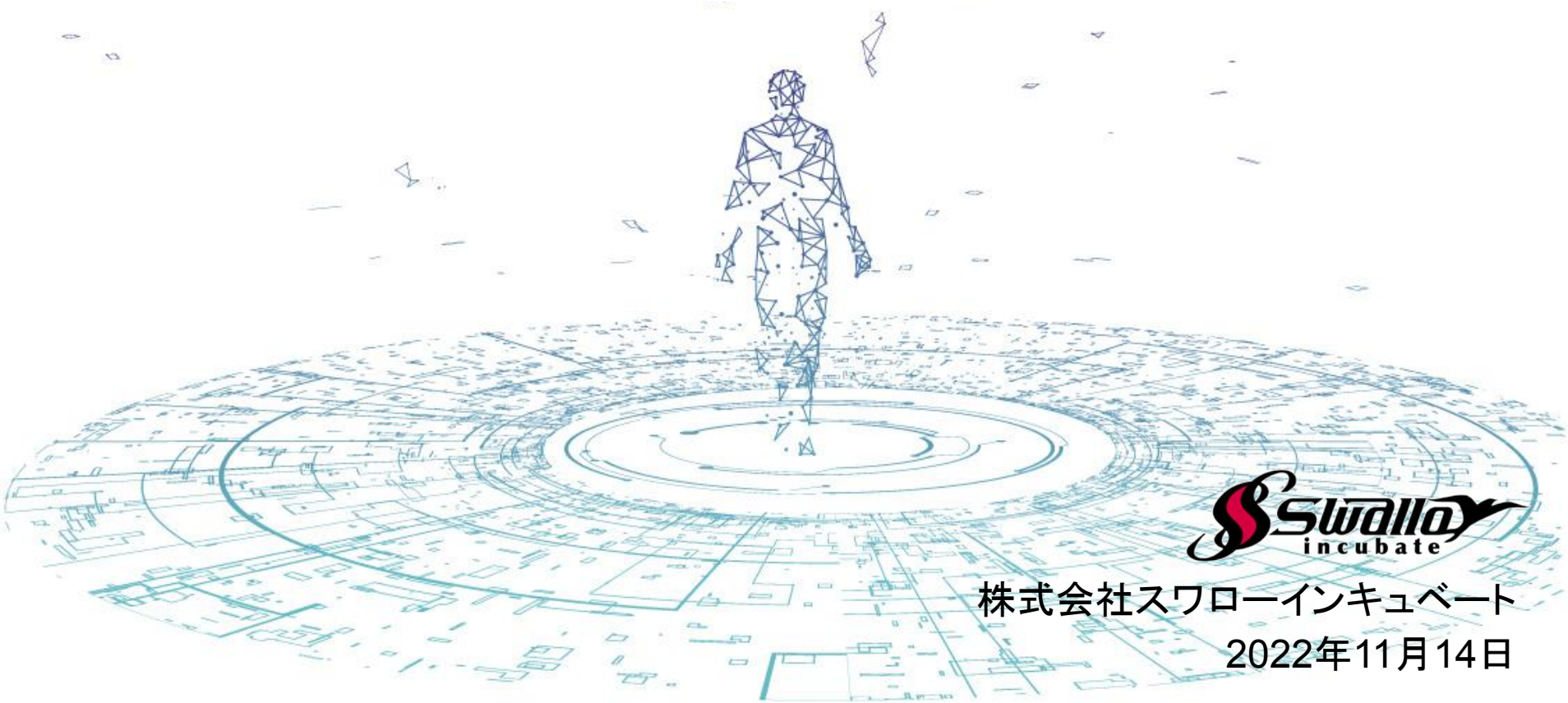
視線検出技術
SDK/インターフェース仕様書(ver.1.6.0)



株式会社スワローインキュベート

2022年11月14日

■はじめに

視線検出SDKは、株式会社スワローインキュベートが提供しています。

本書に基づき、当SDKをご利用いただく前に、以下のご注意事項を十分に読んだ上で、

ご利用いただきますようお願いします。

■ご注意事項

・本書は、予告なしに変更されることがあります。

・本書を無断で、複製、転用、公衆送信、貸与等を行わないようお願いします。

・SDKをご利用いただくには、あらかじめ当社利用規約に同意いただく必要があります。

詳しくは営業担当までお問い合わせください。

お問い合わせ

株式会社スワローインキュベート

視線検出技術　テクニカルサポート窓口

TEL: 029-886-9912　MAIL: support@swallow-incubate.com

■SDK更新履歴

バージョン	日時	変更内容サマリー
ver.1.0.0	2020年01月14日	初版
ver.1.1.0	2020年06月25日	・検出アルゴリズムの改善
ver.1.2.0	2020年09月17日	・検出アルゴリズムの改善
ver.1.3.0	2020年10月23日	・機械学習モデルの導入
ver.1.4.0	2021年02月22日	・検出アルゴリズムの改善
ver.1.5.0	2022年01月15日	・ディープラーニングモデル導入 ・その他微修正
ver.1.6.0（本版）	2022年11月14日	・目位置検出ディープラーニングモデルの導入 ・目の開閉判定精度向上 ・その他微修正

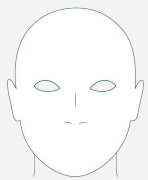
このSDKでできること

■このSDKでできること

視線検出技術SDKでは、ディープラーニングモデルを用いて、顔・目の特徴点などを検出し、それらの情報を元に「顔向き・目向きの検出」や「カメラから目までの視距離を推定する」ことが可能です。これらの検出・判定処理を、WEBカメラなどの可視光RGBカメラを用いてできることが特徴となります。

DL = 深層学習ベース

特徴点検出



DL
顔位置検出

DL
目位置検出

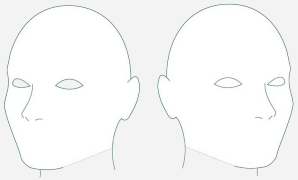
DL
虹彩位置検出

DL
メガネ・サングラス判定

DL
まぶた位置検出

DL
目尻目頭検出

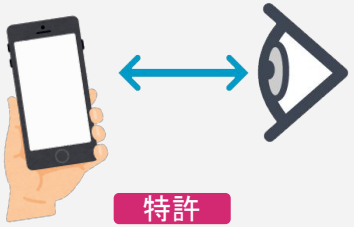
顔向き・目向き検出



DL
顔向き検出

DL
目向き検出

視距離推定



特許
視距離推定

ご利用にあたって

■ご利用環境

現在のバージョンでは、以下のご利用環境に対応しています。

項目	内容	
インターフェース	C++言語 (C++11以降)	
対応OS	Windows OS / Linux OS / macOS / Raspberry Pi OS iOS / Android OS	
CPUアーキテクチャ	64bit系（x86_64 / Arm64） ※一部32bit系にも対応します	
推奨メモリ	2GB以上を推奨	
依存ライブラリ	OpenCV 4.5.5 / OpenCV Contrib 4.5.5	
実行環境 / ビルド環境	Linux kernel 4.9以降	gccを推奨
	macOS 10.11以降	
	Raspbian Buster 推奨	
	Windows 10 以降 64bit系	MicroSoft Visual C++コンパイラを推奨
	iOS 14以降	最新のXcodeの利用を推奨
	Android OS 7.0以降	最新のAndroid Studio / Gradle / NDK の利用を推奨

※その他の環境でのご利用を希望される場合は、お問い合わせください。

■推奨入力画像

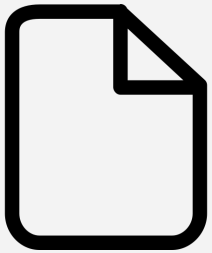
現在のバージョンでは、以下の入力画像を推奨しています。

項目	内容
画像カラー仕様	**RGBカラー画像** (8bit3ch または 8bit4ch) ※RGB565は非対応です
センサ種別	**可視光センサ画像のみ**(赤外線センサ画像には対応していません)
推奨フレームサイズ	**VGA(640 x 480)**サイズ以上
入力画像解像度	<検出される顔領域のピクセル数 > **最低 width 100px 以上**
顔領域の明るさ	顔領域 平均輝度値 **50.0**以上(8bit256階調)
撮影距離	<カメラから顔までの距離 > **～1.5m程度まで** ※入力画像解像度を上げることで距離を伸ばすことも可能です。顔領域最低サイズを参考にしてください。
同時検出可能人数	**1名のみ**

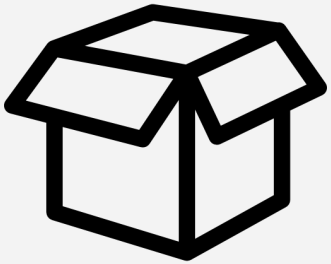
※その他の入力画像でのご利用を希望される場合は、お問い合わせください。

SDK構成

■SDK構成



interfaceファイル
(hpp)



動的リンクライブラリ
(dll/so/dylibなど)

C++コード / ビルド手順

視線検出ライブラリ

視線検出サンプルアプリ

期限付きトークン

USBドングル

アクティベーションツール
(いずれか1つ)

ご利用マニュアル
(本書)

■SDK構成

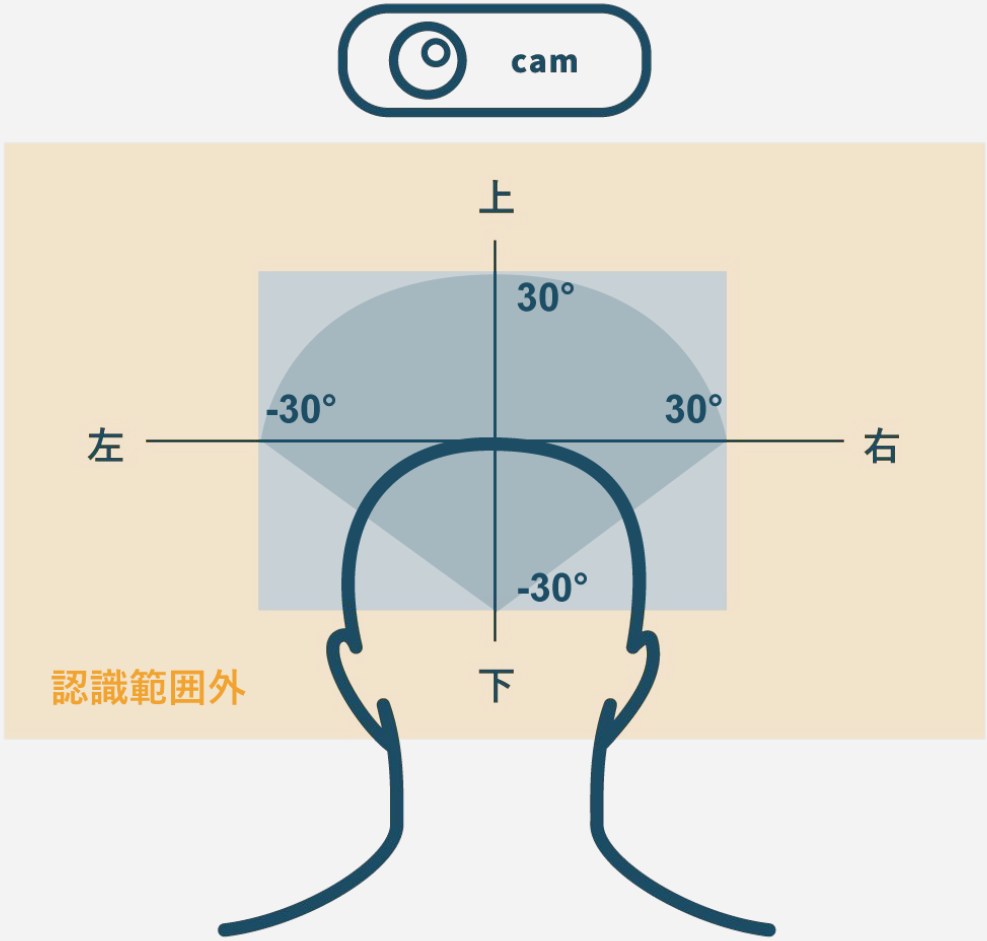
本SDKは動的リンクライブラリとそのインターフェースであるヘッダーファイルで構成されています。C++インターフェースとなっていますが、
スマホOSには、C++言語向けのラッパーサンプルコードを用意しています。

OS	提供物
macOS	EyeTrack.hpp （インターフェースファイル） libEyeTrack.dylib サンプルアプリ（C++）
Windows OS	EyeTrack.hpp EyeTrack.dll (実行時参照ライブラリ) EyeTrack.lib (ビルド時取込ライブラリ) サンプルアプリ（C++）
各種Linux OS **(Raspbian OS含む)**	EyeTrack.hpp（インターフェースファイル） libEyeTrack.so サンプルアプリ（C++）
iOS	EyeTrack.Framework（EyeTrack.hpp含む） サンプルアプリ（Objective-C ラッパーサンプルコード込み） iOS向けOpenCV + OpenCV Contribビルド済ライブラリ
Android OS	EyeTrack.hpp libEyeTrack.so（arm64-v8a / armeabi-v7a / x86 / x86_64） サンプルアプリ（JNI ラッパーサンプルコード込み） Android向けOpenCV + OpenCV Contribビルド済ライブラリ

ライブラリ仕様

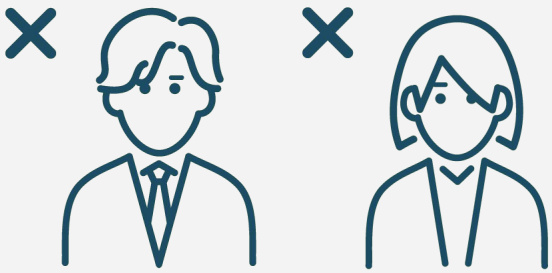
■ライブラリ仕様 - 検出可能画角

現在のバージョンでは、検出可能な画角は、上下左右ともに30°程度ですが
顔の形状などによるため、個人差があります。

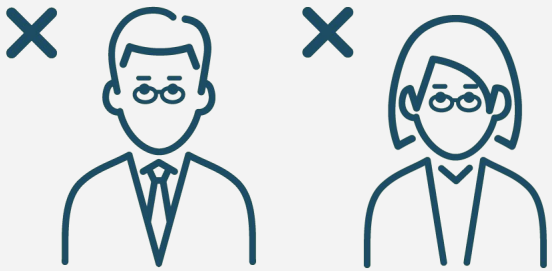


■ライブラリ仕様 - ✕ 検出不可となるケース

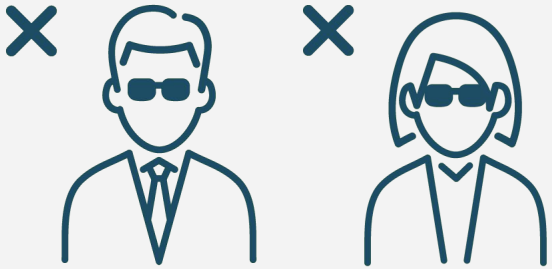
目の位置検出においては、以下のケースで検出エラーになりやすくなりますのでご注意ください。



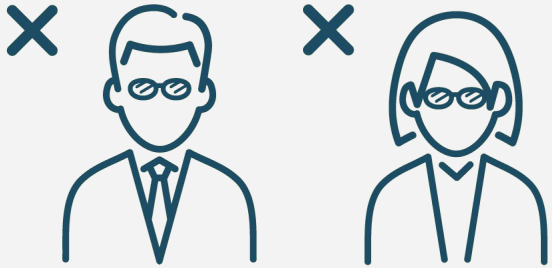
髪の毛が目にかかっている



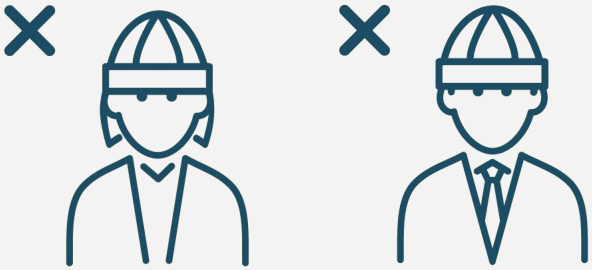
メガネフレームが目にかかっている



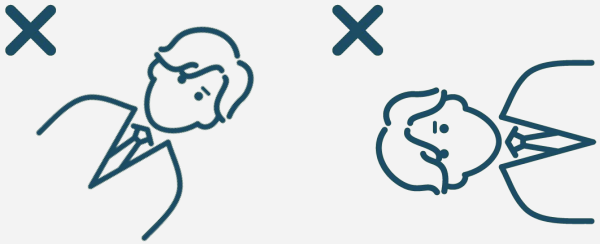
レンズが黒いサングラス



外光の映り込みが激しい



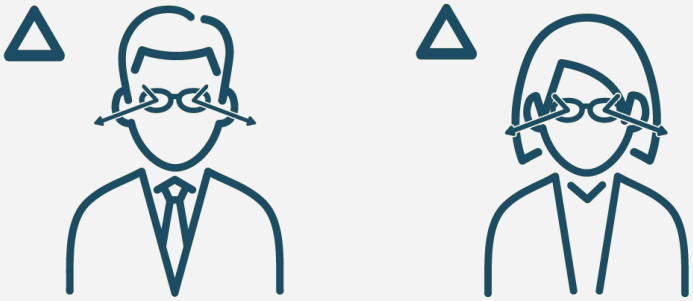
帽子を深くかぶっている



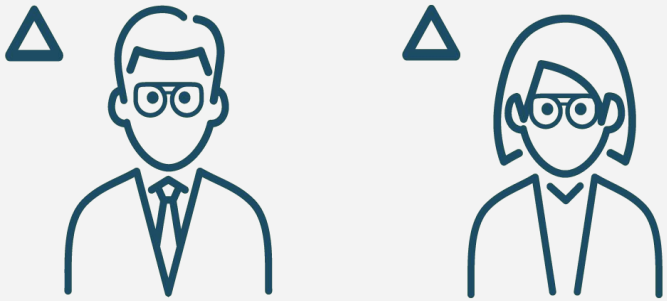
顔が傾いて撮影されている

■ライブラリ仕様 - △ 検出に影響を与える場合があるケース

目の位置検出においては、以下のケースで検出に影響を与えやすくなりますのでご注意ください。



ブルーライトカットメガネ
(ブルーライト反射が強いとエラーになる場合あり)

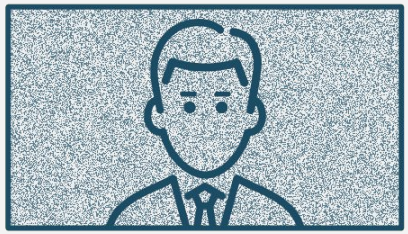


色付き透明サングラス
(色調変化によりエラーとなる場合あり)

■ライブラリ仕様 - △ 検出に影響を与える場合がある撮影状況

現在のバージョンでは、以下の撮影状況で検出エラーとなる場合があります。

△



×露光が不十分
(環境光の差に影響を受けます)

△

×オートフォーカス制御
(フォーカシング中検出エラーになりやすい)

△

×カメラの設置が
まっすぐではない

△

×画角から外れている
(鼻より上しか写っていない)

△

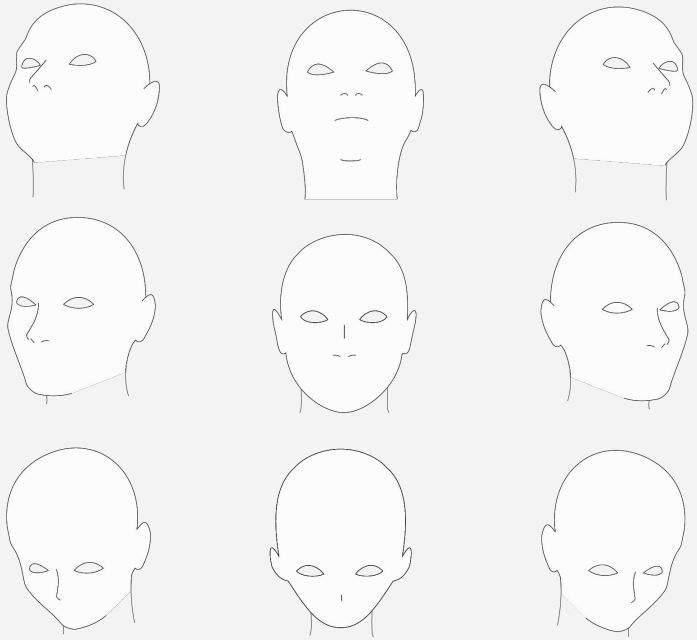
×画角から外れている
(片目しか写っていない)

■ライブラリ仕様 - 機能一覧

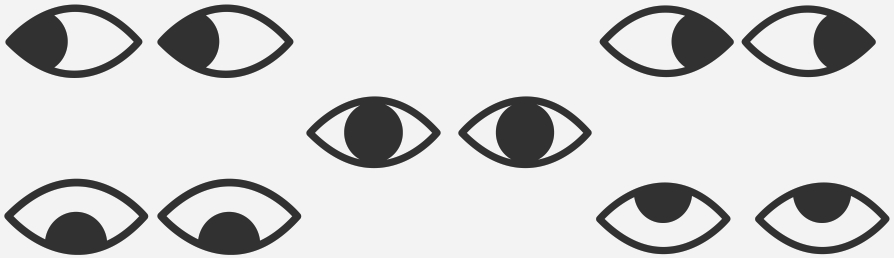
項目	機能項目	要件
基本機能	アクティベーション機能	EyeTrackオブジェクトのプロセス立ち上げごとに、 USBまたはトークンによるアクティベーションを行います。
	モデルファイルの読み込み	弊社で用意したテンプレートファイルを読み込みます。
	パラメータ調整	EyeTrackオブジェクトの初期化時に所定のパラメータ調整を行うことができます。
	画像データの読み込み	EyeTrackクラスにて、画像データを読み込ませることができます。
各種情報 センシング機能	顔検出機能	入力された画像から、深層学習モデルを用いた推論を行い、顔領域候補を検出します。
	複数顔検出機能	入力された画像に複数の顔が検出された場合は、顔横幅サイズが最も大きな顔をメインの顔として処理します。
	顔中心点座標検出機能	検出された顔領域画像から顔中心点座標を検出します。
	顔向き判定機能	検出された顔領域画像から、深層学習モデルを用いた推論を行い、水平方向・垂直方向それぞれの顔向きを判定します。
	メガネ装着判定機能	検出された顔領域画像から、深層学習モデルを用いた推論を行い、メガネの装着の有無及びメガネありの場合は、クリアレンズメガネかサングラスかの区別も可能です。
	目検出機能	検出された顔領域画像の中から、目を検出します。両目を検出できた場合は、左右判定を行います。
	目の向き判定機能	検出された目画像から、深層学習モデルを用いた推論を行い、水平方向・垂直方向それぞれの目の向きを判定します。
	目の開閉判定機能	検出された目画像から、深層学習モデルを用いた推論を行い、目の開閉状態を判定します。
	虹彩(黒目)検出機能	検出された目画像から、虹彩(黒目)位置検出を行います。虹彩検出できた場合は、虹彩中心座標と虹彩半径を取得することができます。
	まぶた・目尻目頭検出機能	検出された目画像から、虹彩中心座標と同じx座標上の上下まぶた位置のxy座標、および目尻目頭の位置のxy座標をそれぞれ検出します。

■ライブラリ仕様 - 機能一覧

項目	機能項目	要件
視線推定基礎情報	顔の左右向き検出	各種センシング情報をもとに、左右の顔の向きを-100～100で定量化します。
	顔の上下向き検出	各種センシング情報をもとに、上下の顔の向きを-100～100で定量化します。
	目の左右向き検出	各種センシング情報をもとに、左右の目の向きを-100～100で定量化します。
	目の上下向き検出	各種センシング情報をもとに、上下の目の向きを-100～100で定量化します。
	視距離推定	各種センシング情報をもとに、カメラから目までの距離を推定します。



顔向き検出



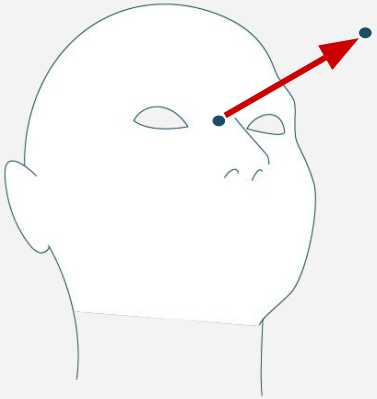
目の向き検出

視距離推定

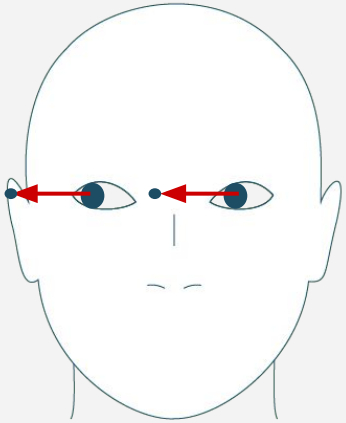
■ライブラリ仕様 - 機能一覧

項目	機能項目	要件
視線向きベクトル	顔向きベクトル	顔向きのベクトル終点座標を取得することが可能です。顔中心点座標と結ぶことで顔向きベクトルを得ることができます。
	左目向きベクトル	左目視線ベクトル終点座標を取得することが可能です。左目(虹彩)中心点座標と結ぶことで左目向きベクトルを得ることができます。
	右目向きベクトル	右目視線ベクトル終点座標を取得することが可能です。右目(虹彩)中心点座標と結ぶことで右目向きベクトルを得ることができます。
視線プロット先推定機能		顔の向きベクトル、目の向きベクトル、視距離推定を基礎情報として、入力フレーム上に、視線先推定プロットを行うことが可能です。

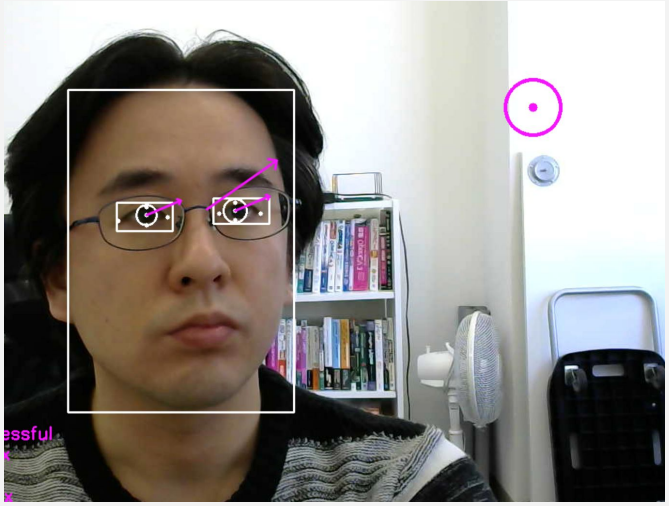
顔向きベクトルは、顔中心点座標から顔向きベクトル終点座標を結ぶ線となり
目向きベクトルは、虹彩中心点座標から目向きベクトル終点座標を結ぶ線で構成されます



顔向きベクトル 例



目向きベクトル 例



■ライブラリ仕様 - 機能一覧

入力画像に対する左右目の判定は、
画像向かって左側の目を左目、画像向かって右側の目を右目として出力します。
上下まぶた、目尻目頭についても以下同様です。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明	
struct **ExternalFilePath**	この構造体について		顔位置検出、顔特徴点の検出を行うためのモデルファイルを読み込ませるために、EyeTrackクラスのオブジェクト化直後に、本構造体をinit()関数の引数として与えます。	
	顔検出用	std::string	faceDetectModel	顔位置検出に用いるモデルファイルを読み込む変数です。
		std::string	faceDetectParam	
	顔向き判定用	std::string	faceDirHModel	顔向き判定に用いるモデルファイルを読み込む変数です。
		std::string	faceDirVModel	
	メガネ判定用	std::string	glassesModel	メガネ判定に用いるモデルファイルを読み込む変数です。
	目検出用	std::string	eyeDetectModel	目検出に用いるモデルファイルを読み込む変数です。
	目の開閉判定用	std::string	eyeStatusModel	目の開閉判定に用いるモデルファイルを読み込む変数です。
	目向き判定用	std::string	eyeDirHModel	目の向き判定に用いるモデルファイルを読み込む変数です。
		std::string	eyeDirVModel	
	目ランドマーク検出用	std::string	eyeLandmarkModel	目のランドマークを検出するためのモデルファイルを読み込む変数です。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
struct **Params**	この構造体について		顔位置検出、顔特徴点の検出を行うためのパラメータ値を読み込ませるために、EyeTrackクラスのオブジェクト化直後に、本構造体をinit()関数の第二引数として与えます。本引数は省略可能で、省略した場合は、内部であらかじめ定められた初期設定値で処理が行われます。
	int	minFaceWidth	顔検出を行う最小顔横幅サイズ(px)です。
	int	maxFaceWidth	顔検出を行う最大顔横幅サイズ(px)です。
	float	calibParam	カメラのレンズ、画角に応じて調整する視距離推定パラメータです。
	float	vdParam	視距離推定結果にmm単位で加減算する視距離推定パラメータです。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeData**	このクラスについて		視線推定結果格納クラスです。途中での検出エラー時でも、検出できた部分までのメンバは取得可能です。
	cv::Mat	originalImg	入力した元画像を取得可能です。
	cv::Mat	checkImg	入力した元画像に、検出結果を描画プロットした画像です。
	cv::Mat	faceRectImg	入力した元画像から検出された顔画像を切り出した画像を取得可能です。
	cv::Rect	faceRectArea	検出された顔矩形領域情報を取得可能です。(x座標,y座標,width,height)
	float	faceBrightness	検出された顔矩形領域の平均輝度値を8bit256階調で取得可能です。
	bool	isValidFace	検出された顔において顔の明るさ・サイズが適正かどうかを判定します。
	cv::Point	faceCenter	顔矩形領域において、顔中心位置となるx, y座標を取得可能です。
	std::vector<cv::Rect>	vFaceRect	検出したすべての顔領域情報を取得可能です。(x座標,y座標,width,height)
	short	eyeGlassStatus	検出された顔をもとにした、深層学習モデルによるメガネ装着の有無の推論結果です。 初期値は-1となっており、判定が行われると出力は0～2の整数値となります。 0は裸眼、1はクリアレンズメガネ、2はサングラスであると推定されます。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeData**	cv::Mat	eyeRectImgLeft	検出された左目画像を切り出した画像を取得可能です。
	cv::Mat	eyeRectImgRight	検出された右目画像を切り出した画像を取得可能です。
	cv::Rect	eyeRectAreaLeft	検出された左目矩形領域情報を取得可能です。(x座標,y座標,width,height)
	cv::Rect	eyeRectAreaRight	検出された右目矩形領域情報を取得可能です。(x座標,y座標,width,height)
	float	eyeRectAvgBrightLeft	検出された左目矩形領域の平均輝度値を8bit256階調で取得可能です。
	float	eyeRectAvgBrightRight	検出された右目矩形領域の平均輝度値を8bit256階調で取得可能です。
	short	eyeStatusLeft	検出された顔をもとにした、深層学習モデルによる目の開閉状態の推論結果です。 初期値は-1となっており、判定が行われると出力は0～2の整数値となります。 0は判定エラー、1は閉じ目、2は開き目であると推定されます。
	short	eyeStatusRight	

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeData**	short	irisRadiusLeft	検出された左目瞳(虹彩)半径情報を取得可能です。
	short	irisRadiusRight	検出された右目瞳(虹彩)半径情報を取得可能です。
	cv::Point	irisCenterLeft	検出された左目瞳(虹彩)中心位置情報を取得可能です。
	cv::Point	irisCenterRight	検出された右目瞳(虹彩)中心位置情報を取得可能です。
	cv::Point	eyeLeftUpperEyelid	検出された左目 上まぶた位置情報を取得可能です。
	cv::Point	eyeLeftLowerEyelid	検出された左目 下まぶた位置情報を取得可能です。
	cv::Point	eyeRightUpperEyelid	検出された右目 上まぶた位置情報を取得可能です。
	cv::Point	eyeRightLowerEyelid	検出された右目 下まぶた位置情報を取得可能です。
	cv::Point	eyeLeftCornerL	検出された左目目尻位置情報を取得可能です。
	cv::Point	eyeLeftCornerR	検出された左目目頭位置情報を取得可能です。
	cv::Point	eyeRightCornerL	検出された右目目頭位置情報を取得可能です。
	cv::Point	eyeRightCornerR	検出された右目目尻位置情報を取得可能です。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeData**	float	viewingDistance	カメラから目までの距離（単位：mm）を取得可能です。
	float	faceDirectionH	水平方向の顔向き具合を表すパラメータです。正面を0とし、顔の傾き具合に応じて、左に−100.0～0まで、右に0～100.0の値を返します。
	float	faceDirectionV	垂直方向の顔向き具合を表すパラメータです。正面を0とし、顔の傾き具合に応じて、下に−100.0～0まで、上に0～100.0の値を返します。
	float	eyeDirectionH	水平方向の目の向き具合を表すパラメータです。正面を0とし、目の向き具合に応じて、左に−100.0～0まで、右に0～100.0の値を返します。
	float	eyeDirectionV	垂直方向の目の向き具合を表すパラメータです。正面を0とし、目の向き具合に応じて、下に−100.0～0まで、上に0～100.0の値を返します。
	cv::Point	viewingFaceVector	顔向きベクトルの終点座標を取得可能です。顔中心点座標である cv::Point faceCenterと結ぶことでベクトルを得ることができます。
	cv::Point	viewingEyeVectorLeft	左目の向きベクトルの終点座標を取得可能です。左目虹彩点座標である cv::Point irisCenterLeftと結ぶことでベクトルを得ることができます。
	cv::Point	viewingEyeVectorRight	右目の向きベクトルの終点座標を取得可能です。右目虹彩点座標である cv::Point irisCenterRightと結ぶことでベクトルを得ることができます。
	short	viewingArea	入力画像を均等に9分割した場合の、視線先推定面の値を取得することができます。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeData**	short	faceDirStatusH	水平方向の顔向き判定結果 0=正面 / 1=左向き / 2=右向き / -1= 初期値
	short	faceDirStatusV	垂直方向の顔向き判定結果 0=正面 / 1=上向き / 2=下向き / -1= 初期値
	short	eyeLeftDirStatusH	水平方向の左目向き判定結果 0=正面 / 1=左向き / 2=右向き / -1= 初期値
	short	eyeLeftDirStatusV	垂直方向の左目向き判定結果 0=正面 / 1=上向き / 2=下向き / -1= 初期値
	short	eyeRightDirStatusH	水平方向の右目向き判定結果 0=正面 / 1=左向き / 2=右向き / -1= 初期値
	short	eyeRightDirStatusV	垂直方向の右目向き判定結果 0=正面 / 1=上向き / 2=下向き / -1= 初期値
	std::string	msg	処理結果メッセージを取得することができます。
	void	clear()	EyeDataクラスの全メンバ変数を初期化します。

■ライブラリ仕様 - クラス構成

クラス名	データ型	メンバ名	説明
class **EyeTrack**	このクラスについて		視線検出に必要な顔・目周辺情報を検出するための実行クラスとなります。
	bool	init	初期化を行うメンバ関数です。引数にExternalFilePath構造体をとります。 第二引数にParams構造体をとりますが、省略可能です。
	bool	registerEyeData	メモリ確保したEyeDataクラスを本クラスに登録するための関数です。
	bool	process	cv::Matを引数とし、顔・目周辺情報検出を行うクラスです。 検出結果はすべてEyeDataクラスに格納されます。
	cv::Point	gazePlot	対象となるフレーム、視距離を入力すると、視線プロット先を推定し、座標を取得することが可能です。第五引数より直接プロットも可能です。

■ライブラリ仕様 - クラス構成

◆ USBアクティベーション方式

クラス名	データ型	メンバ名	説明
class **ActivationChallenge**	このクラスについて		EyeTrackクラスを実行する前にアクティベーションを行うための クラスとなります。
	std::string	config()	アクティベーション情報および本ライブラリの情報を取得可能です。
	bool	checkUSB()	USBドングルがマシンに接続されているかをチェックするための関数です。
	bool	validUSB()	USBドングルとライブラリに埋め込まれたIDとが一致するかをチェックするための関数です。本関数からtrueが返ることによりEyeTrackクラスの初期化処理であるinit()関数が成功するようになります。

◆ トークンアクティベーション方式

クラス名	データ型	メンバ名	説明
class **ActivationChallenge**	std::string	config()	アクティベーション情報および本ライブラリの情報を取得可能です。
	bool	checkToken()	本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応トークンとマッチするかどうかのみを判定します。 **本関数からtrueが返るだけではアクティベーションは完了しません。**
	bool	validToken()	本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応するトークン情報を読み取り、有効期限内かどうかをチェックするための関数です。有効期限内であれば本関数からtrueが返り、EyeTrackクラスの初期化処理であるinit()関数が成功するようになります。
	long	getExpDate()	本関数の引数にとるトークン文字列が、ライブラリに埋め込まれた対応するトークン情報を読み取り、有効期限をlong型の数値で返します。例えば返り値が「20221231」であれば、2022年12月31日まで有効なトークンです。

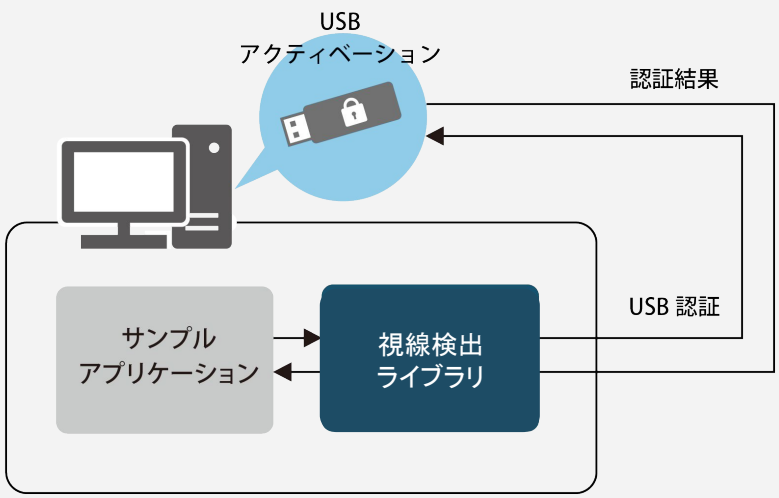
■ライブラリ仕様 - ファクトリ関数

返り値	関数名	説明
EyeTrack*	newEyeTrack()	EyeTrackクラスをオブジェクト化させるための関数です。 内部で派生クラスの生成などが行われるため、EyeTrackクラスのオブジェクト化は、必ずこの関数を用いて行ってください。
void	releaseEyeTrack()	EyeTrackオブジェクトをメモリ開放させるための関数です。 EyeTrackオブジェクトが不使用になった場合は、必ずこの関数を実行してください。
ActivationChallenge*	newActivationChallenge()	ActivationChallengeクラスをオブジェクト化させるための関数です。 内部で派生クラスの生成などが行われるため、ActivationChallengeクラスのオブジェクト化は、必ずこの関数を用いて行ってください。
void	releaseActivationChallenge()	ActivationChallengeオブジェクトをメモリ開放させるための関数です。 ActivationChallengeオブジェクトが不使用になった場合は、必ずこの関数を実行してください。

■ライブラリ仕様 - アクティベーション

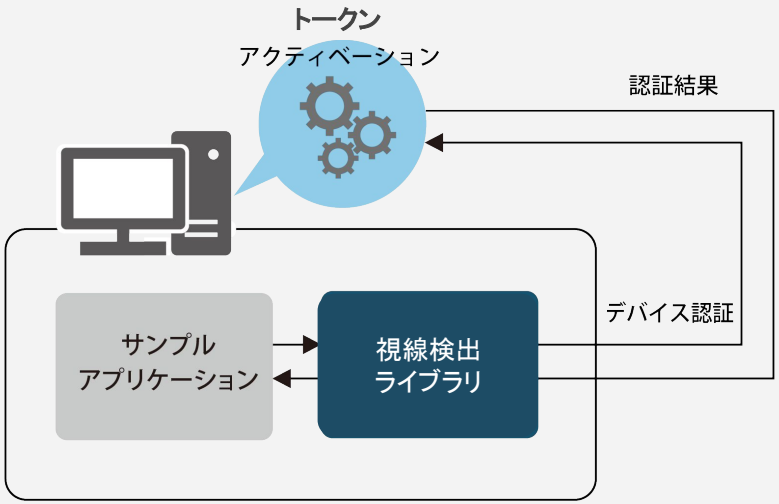
開発版ライセンスでは、本ライブラリをご利用いただくにあたって、アクティベーションが必要となります。アクティベーションには、USB方式と有効期限を利用したトークン方式の2タイプがあります。

◆ USBアクティベーション



有償（USBドングル5本まで込み）
※PC版は原則こちらでお願いしております

◆ トークンアクティベーション



無償
※主にスマホ/タブレット端末で利用企業様向けとなります

■ライブラリ仕様 - アクティベーション情報

ActivationChallengeオブジェクトのconfig()を使用することで、返り値に、
アクティベーション情報やライブラリ基本情報を取得することができます。
弊社にお問い合わせいただく場合に取得をお願いする場合があります。

◆ USBアクティベーション版 config() 実行結果例

```
[toshikazuohno@sample]$ ./get_config
Activation   : USB
Client ID    : 2349799210
Client Name  : Swallow Incubate
SDK Version  : EyeTrackSDK - ver.1.1.1
Start-Date   : 2020-06-25
[toshikazuohno@sample]$
```

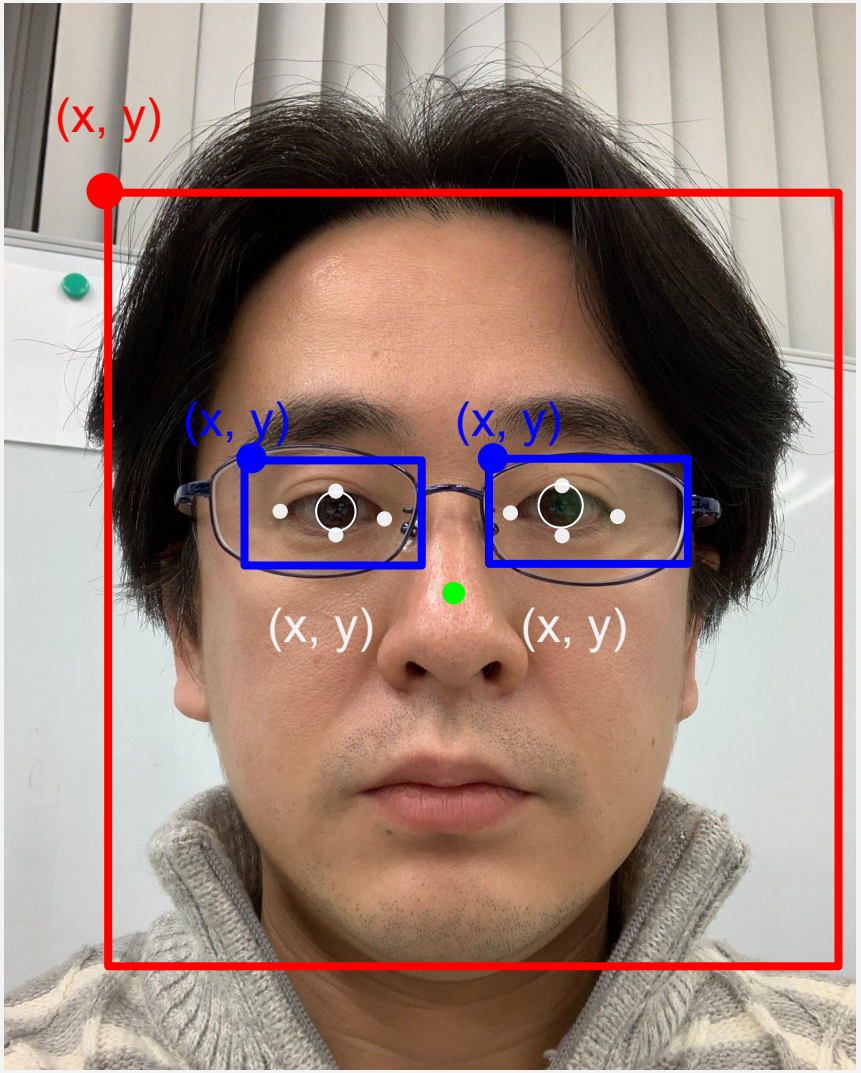
◆ トークンアクティベーション版 config() 実行結果例

```
[toshikazuohno@sample]$ ./get_config
Activation   : Token
Client ID    : 2349799210
Client Name  : Swallow Incubate
SDK Version  : EyeTrackSDK - ver.1.1.1
Start-Date   : 2020-06-25
[toshikazuohno@sample]$
```

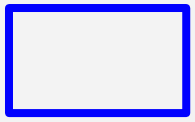
出力データ詳細

■出力データ図解

EyeData型より取得できる検出位置データの値は、以下の通りとなります。
cv::Rect型のxy座標は、矩形領域の左上頂点座標となります。

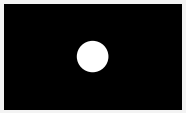


= faceRectArea
(x, y, width, height)



= eyeRectArea
(x, y, width, height)

= irisDiameter(irisRadius x 2)



= upperEyelid / lowerEyelid
eyeCornerL / eyeCornerR
(x, y, width, height)

= faceCenter

■message一覧

EyeData型のメンバ変数であるmsgにて取得できる主なメッセージは以下のいずれかとなります。その他のエラーが発生した場合はお問合せください。

メッセージ内容	内容
Process Successful	全ての処理が完了しました
Error: init() first	process()実行前に、init()を実行してください
Error: registerEyeData() first	process()実行前に、registerEyeData()を実行してください
Error: Empty input image	入力画像が存在しません
Error: Input RGB image	RGB画像を入力してください
Error: Failed to detect face	顔検出に失敗しました（顔が存在しない）
Error: Lack of face area brightness	顔領域の明るさ不足です（顔領域平均輝度値45以上 – 8bit256階調）
Error: Failed to detect eyes	目検出に失敗しました
Error: Failed to detect open eye	開き目が存在しません
Error: Failed to detect eye landmark points	両目とも目のランドマーク検出に失敗しました

視線検出基礎情報について

■視線検出基礎情報 - メガネ判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、メガネ装着の有無をディープラーニングを用いて判定します。処理結果が格納EyeDataクラスの、eyeGlassStatusメンバより0～2のいずれかの値が出力されることによりメガネ装着判定を行うことが可能です。サングラスであると判定された場合は、目検出以降の目の特徴点検出処理は行いません。

裸眼

eyeGlassStatus = 0

クリアレンズ
メガネ

eyeGlassStatus = 1

サングラス

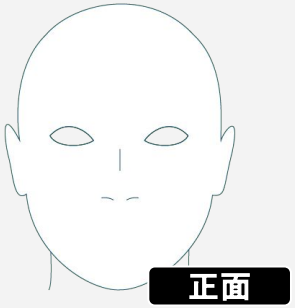
eyeGlassStatus = 2

■視線検出基礎情報 - 顔向き判定

ディープラーニング版

単一フレームの画像から、顔が検出された場合、顔向きをディープラーニングを用いて判定します。処理結果が格納EyeDataクラスの、faceDirStatusH、faceDirStatusVメンバより0～2のいずれかの値が出力されることにより顔向きの判定を行うことが可能です。

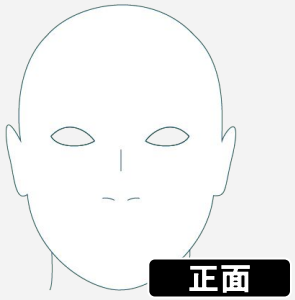
faceDirStatusH = 1



faceDirStatusH = 0

faceDirStatusH = 2

faceDirStatusV = 1



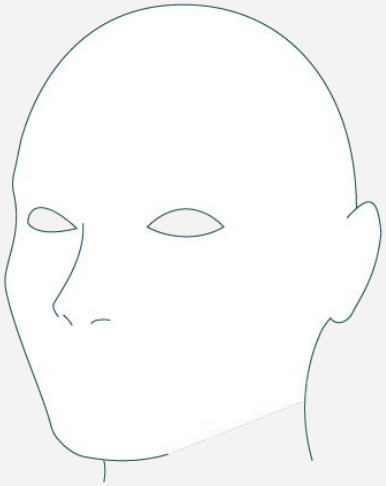
faceDirStatusV = 0

faceDirStatusV = 2

■視線検出基礎情報 - 水平方向の顔向き度

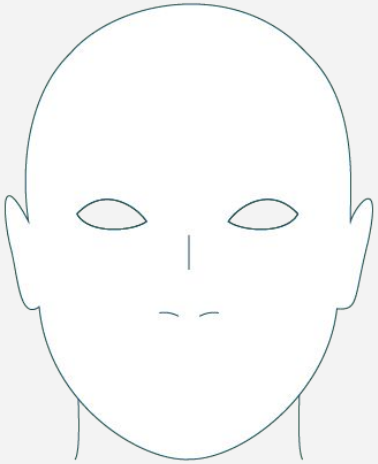
特許取得済

左右の顔の向きは、EyeDataクラスのメンバ変数であるfaceDirectionHより取得できます。この値は、カメラに対して正面を0として、顔向きの変化に応じて左側へ 0 ～ -100.0(約-30°)、右側へ0 ～ 100.0(約+30°)の間で定量化されます。
検出限界角度は、左右それぞれ約30度程度ですが個人差があります。



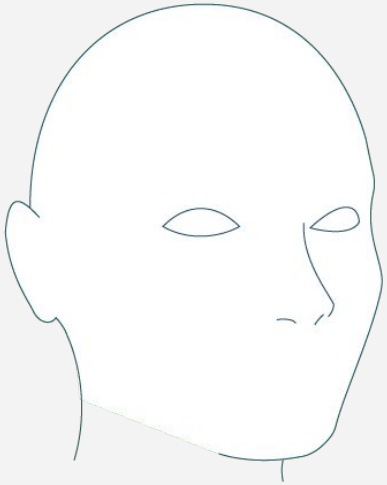
左限値

faceDirectionH
-100.0



正面

faceDirectionH
0.0



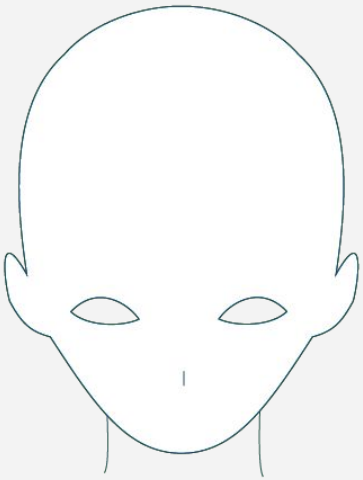
右限値

faceDirectionH
100.0

■視線検出基礎情報 - 垂直方向の顔向き度

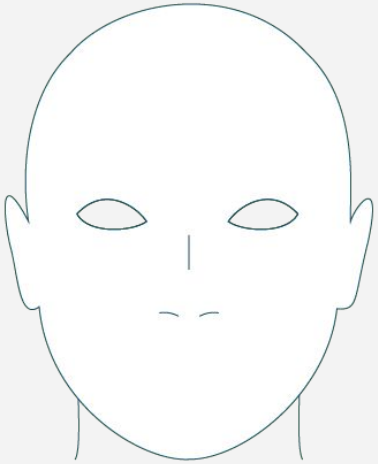
特許取得済

上下の顔の向きは、EyeDataクラスのメンバ変数であるfaceDirectionVより取得できます。この値は、カメラに対して正面を0として、顔向きの変化に応じて下側へ 0 ～ -100.0、上側へ0 ～ 100.0の間で定量化されます。
検出限界角度は、個人差があります。



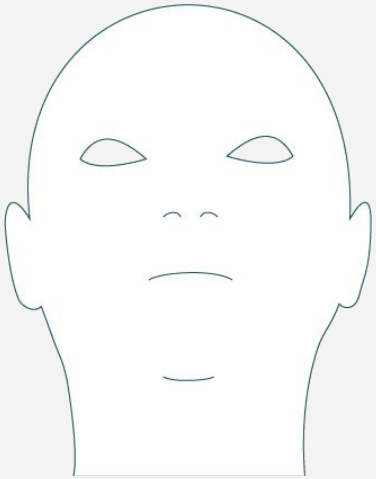
下限値

faceDirectionV
-100.0



正面

faceDirectionV
0.0



上限値

faceDirectionV
100.0

■視線検出基礎情報 - 目向き検出

ディープラーニング版

目の向きは、EyeDataクラスのメンバ変数であるeye(Left/Right)DirStatusHおよびeye(Left/Right)DirStatusVより取得できます。値はshort型の0～2のいずれかが返ります。値の詳細は以下の通りとなります。

左向き
eye(Left/Right)DirStatusH
1

中央
eye(Left/Right)DirStatusH
0

右向き
eye(Left/Right)DirStatusH
2

下向き
eye(Left/Right)DirStatusV
2

中央
eye(Left/Right)DirStatusV
0

上向き
eye(Left/Right)DirStatusV
1

■視線検出基礎情報 - 目向き度

特許取得済

目の向きは、EyeDataクラスのメンバ変数であるeyeDirectionHおよびeyeDirectionVより取得できます。それぞれの値は、カメラに対して正面を0として、目の上下左右への向き具体の変化に応じて左側・下側へそれぞれ -100.0～0、右側・上側へそれぞれ0 ～ 100.0の間で定量化されます。

左限値
eyeDirectionH
-100.0

中央値
eyeDirectionH
0.0

右限値
eyeDirectionH
100.0

下限値
eyeDirectionV
-100.0

中央値
eyeDirectionV
0.0

上限値
eyeDirectionV
100.0

■視線検出基礎情報 - 目の開閉判定

ディープラーニング版

目の開閉パラメータは、EyeDataクラスのメンバ変数であるeyeStatus(Left/Right)より取得できます。値はshort型の0～2のいずれかが返ります。値の詳細は以下の通りとなります。

eyeStatus = 0

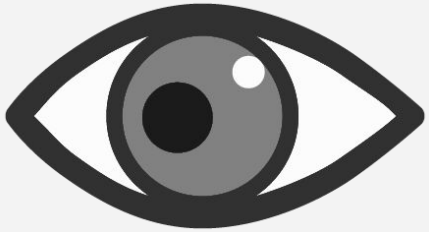
Error

目検出エラー

eyeStatus = 1

目が閉じている

eyeStatus = 2

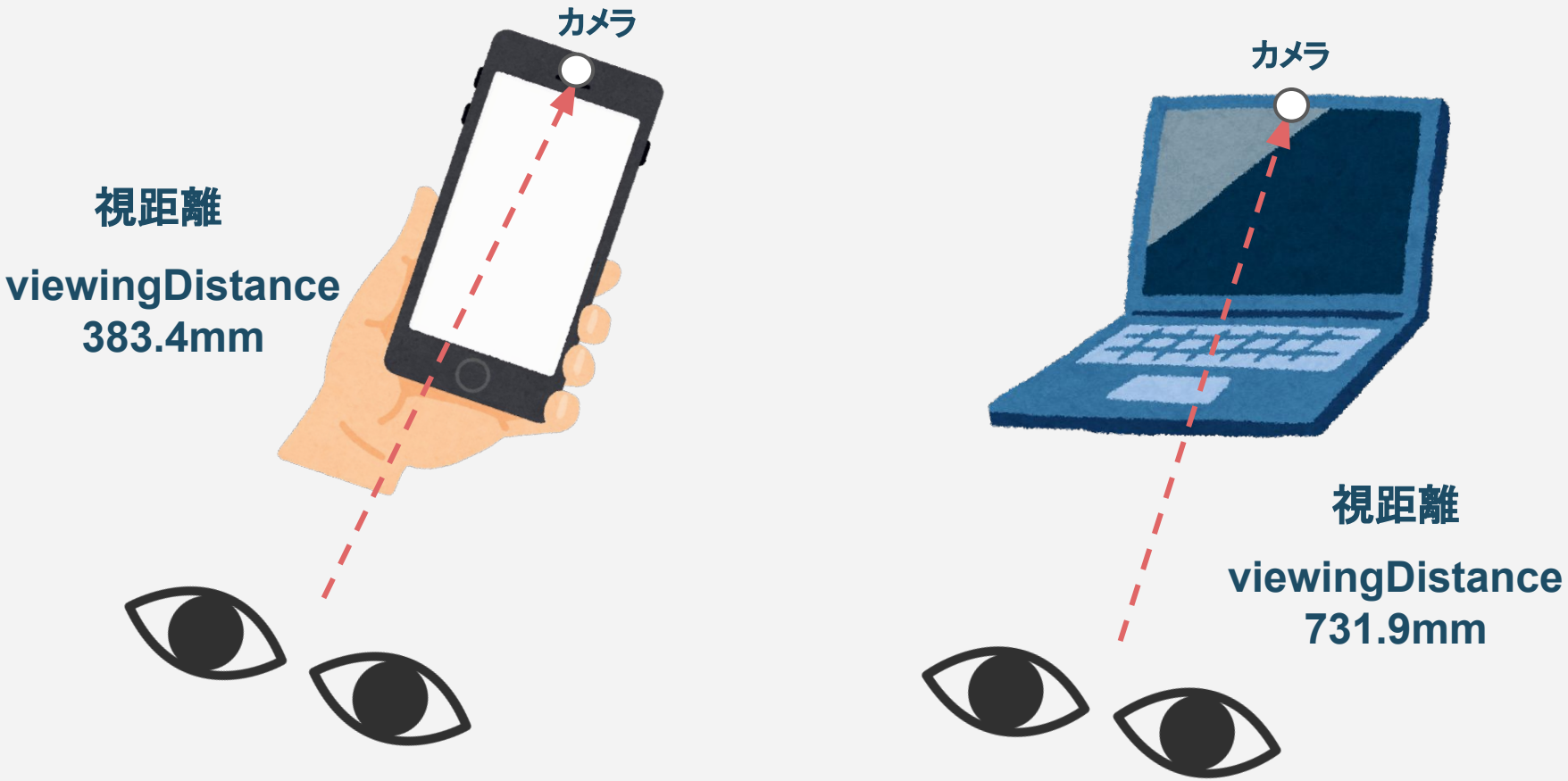


目が開いている

■視線検出基礎情報 - 視距離推定

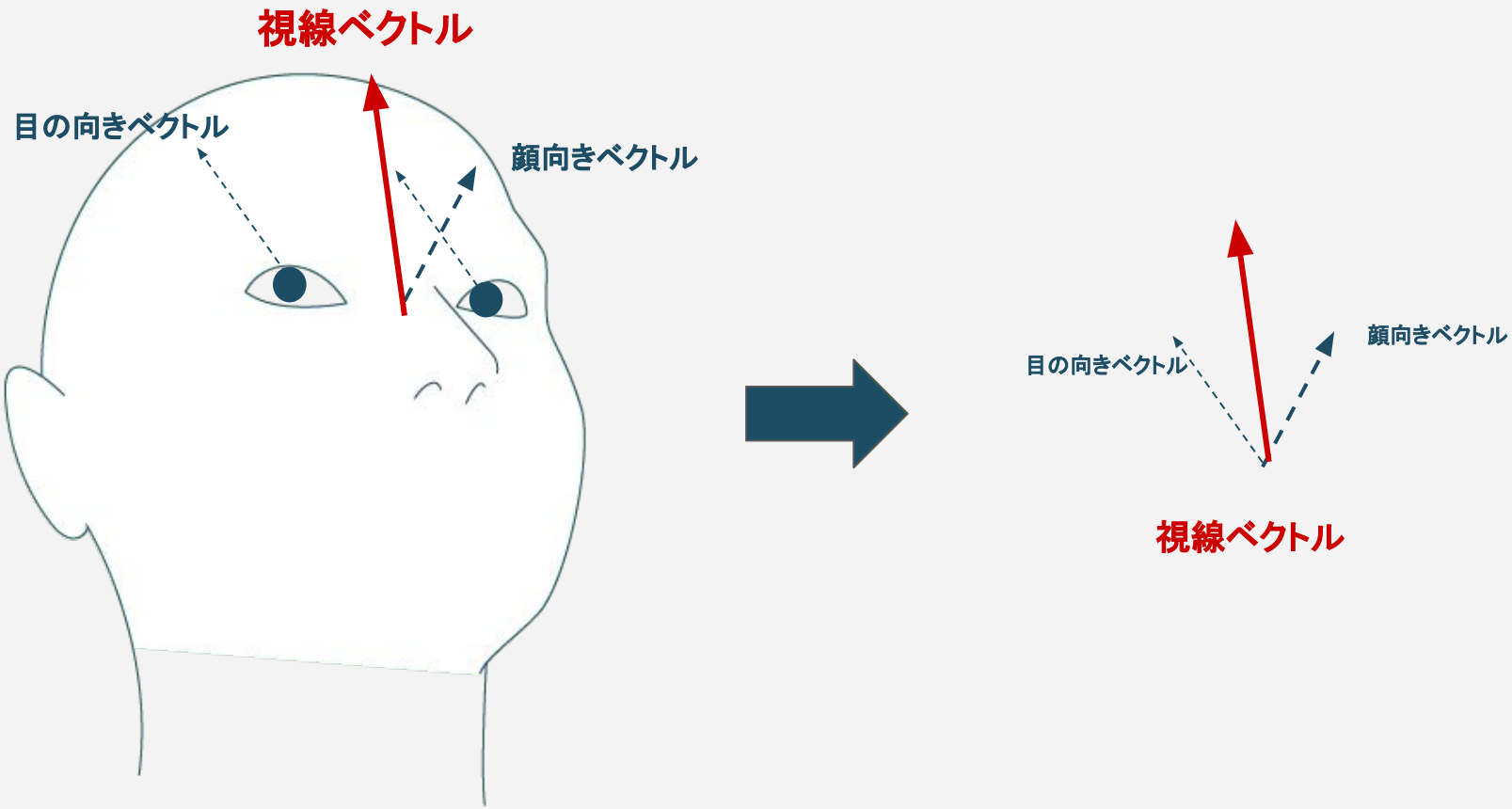
特許取得済

目検出結果をもとに、カメラから目までの距離を取得できます。
視距離の値は、EyeDataクラスのメンバ変数であるviewingDistanceより取得することができ、単位mm（ミリメートル）で取得できます。
単位cm（センチメートル）にする場合は、この値を10で割ってください。



■視線検出基礎情報 - 視線ベクトル

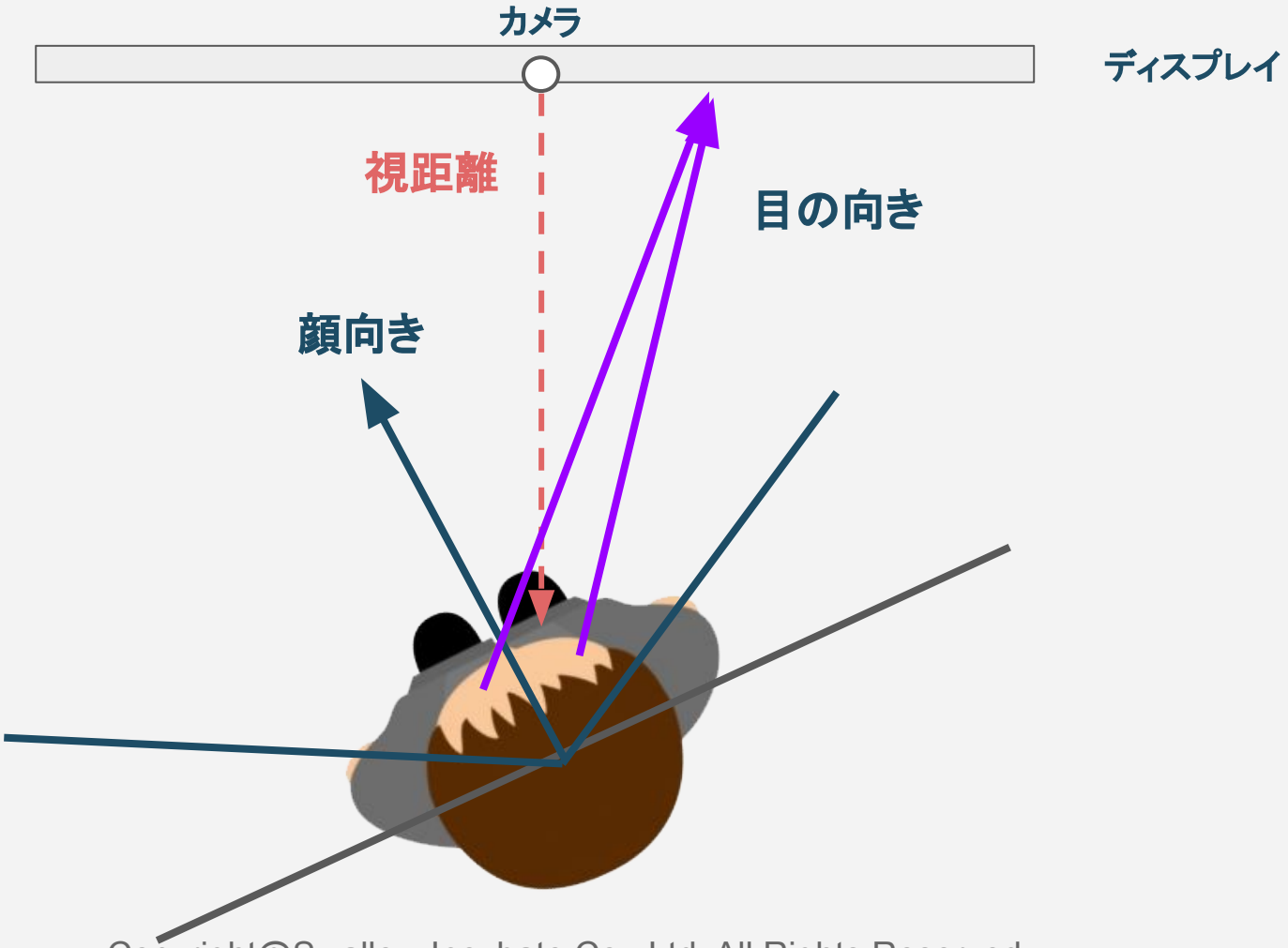
自身で視線ベクトルを得る場合は、ライブラリより取得可能な顔向きベクトルと、目の向きベクトルから合成ベクトルを生成、重みづけなどを行うことで得ることが可能です。以下は、視線ベクトルを得る例です。



※目の向きベクトルは、左右同じ量となります。

■視線検出基礎情報 - 視線プロット先推定

視線先の推定は、顔向きベクトルおよび目の向きベクトルを用いた視線プロット先計算を、EyeTrack::gazePlot()にて行うことが可能です。
その他、お客様にて、ライブラリより取得した顔向き、目の向き、視距離をもとに視線プロット先を推定計算していただくことも可能です。



■視線検出基礎情報 - 注視エリアの推定

視線先のおおまかな領域は、EyeDataクラスのメンバ変数であるviewingAreaより取得できます。値はshort型の0～9のいずれかが返ります。入力フレームを均等に9分割した際の視線推定先領域を以下の値の通り示します。

viewingArea = 0

Error

viewingArea = 1～9

3	4	5
2	1	6
9	8	7

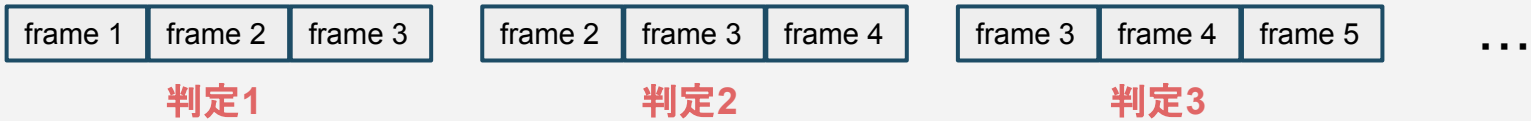
まばたき判定の例

■まばたき判定 - サンプル

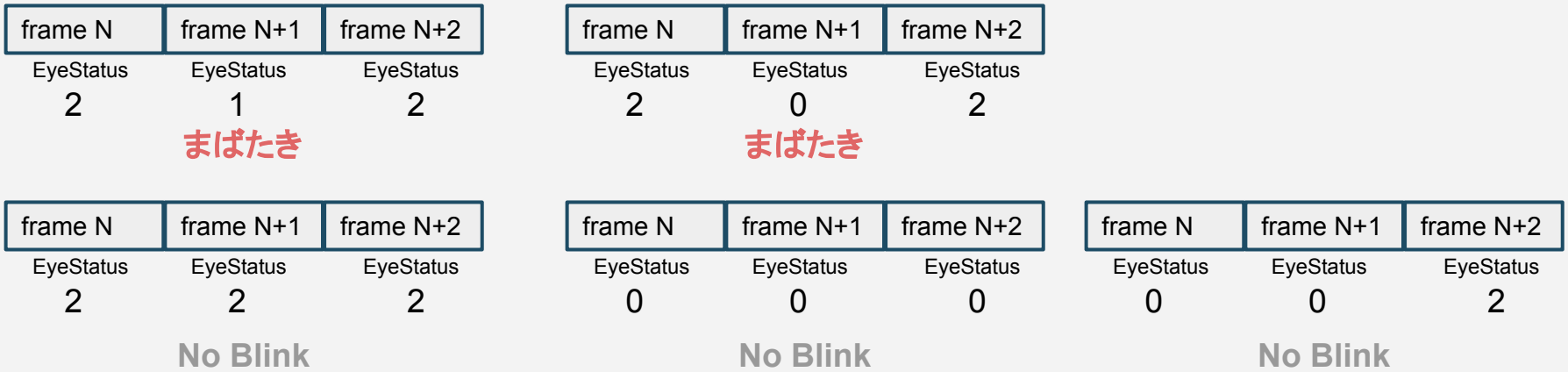
目検出ライブラリは、1フレームの検出結果を返すのみとなりますので、まばたき判定は、検出結果の時間変化をもとに独自で判定していただく必要があります。以下にまばたき判定の例を掲載いたしますが、独自に判定アルゴリズムを策定いただくことも可能です。詳しくはサンプルアプリを参照してください。

<まばたき判定に用いるフレーム数 > - 3つずつ

<まばたき判定に使う時間フレーム > 連続する3フレーム



<まばたき判定アルゴリズム >

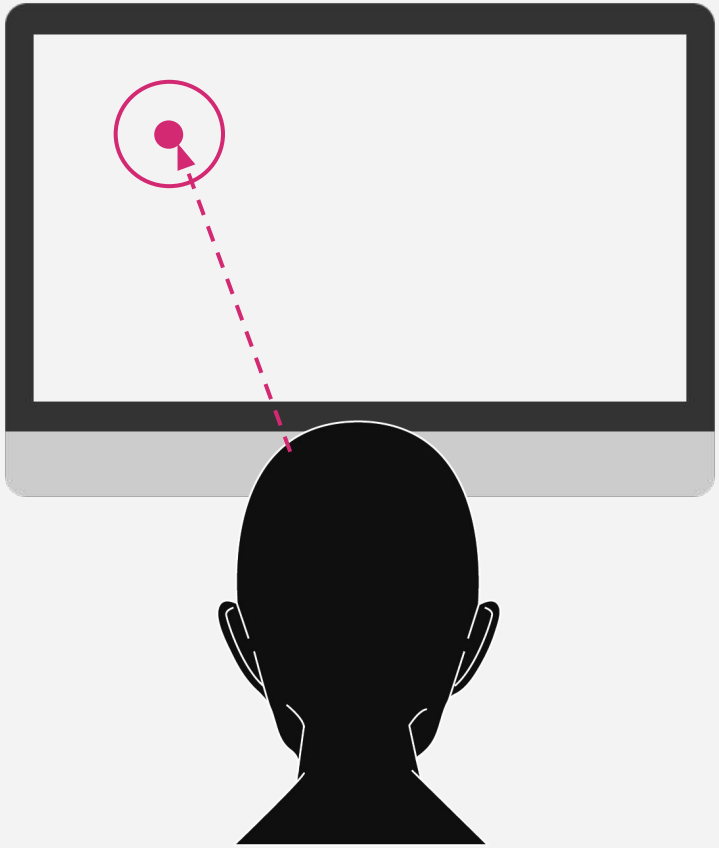


視線UIへの応用

■ 視線検出技術 – 視線UIとしての使用

視線検出技術をユーザーインターフェースとしてご利用いただく場合は、
視線プロット先をマウスの代替とし、まばたきやウィンク検出を、クリック操作として代替していただくことが可能です。

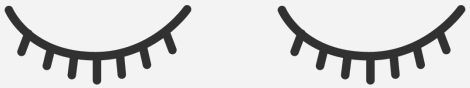
画面プロット例



操作例

左クリック

右クリック



ダブルクリック

サンプル

https://youtu.be/smClTfNdvto