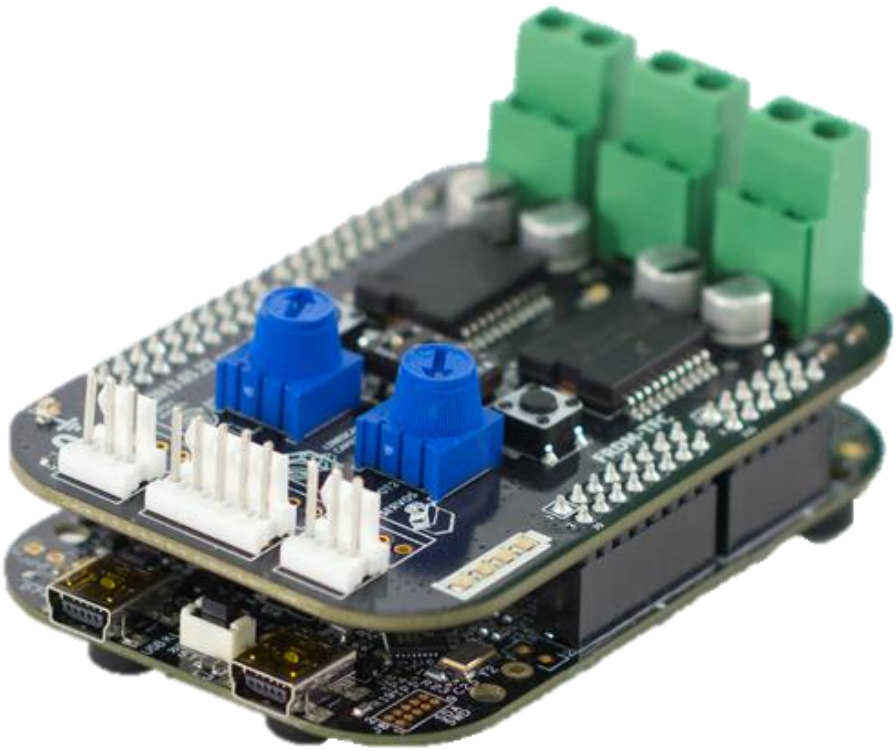
A simple way to detect the track's line for the Freescale Cup Smart Race

By: Technical Information Center

Introduction

This document explains how to detect the track's line for the Freescale Cup smart race from data acquired with the linear camera with the FRDM-KL25Z as is described in the document "Line scan camera with KSDK [ADC + PIT + GPIO]".

Detect the track's line is very important because it involves to know the path that the car must follow.



freescale semiconductor

Contents

1. About this document

This document is focused in KSDK 1.2.0 with KDS 3.0.0, the document "Create a new KSDK 1.2.0 project in KDS 3.0.0" explains with detail the way to create a new KSDK project.

This document is a continuation of the documents "Line scan camera with KSDK [ADC + PIT + GPIO]" and "Controlling speed in DC motors and position in servomotors with the FRDM-KL25Z and the Kinetis SDK [FTM + GPIO]" where the line scan camera, the servo motors, the DC motors and the FRDM-TFC shield were enabled. Now, it is time to process the data provided by the linear camera.

Figure 1. Track for the Freescale Smart Car Race.

This document enables the remaining block before to build the application. If it is desired to use the smart care in a different track, just this algorithm must be changed.

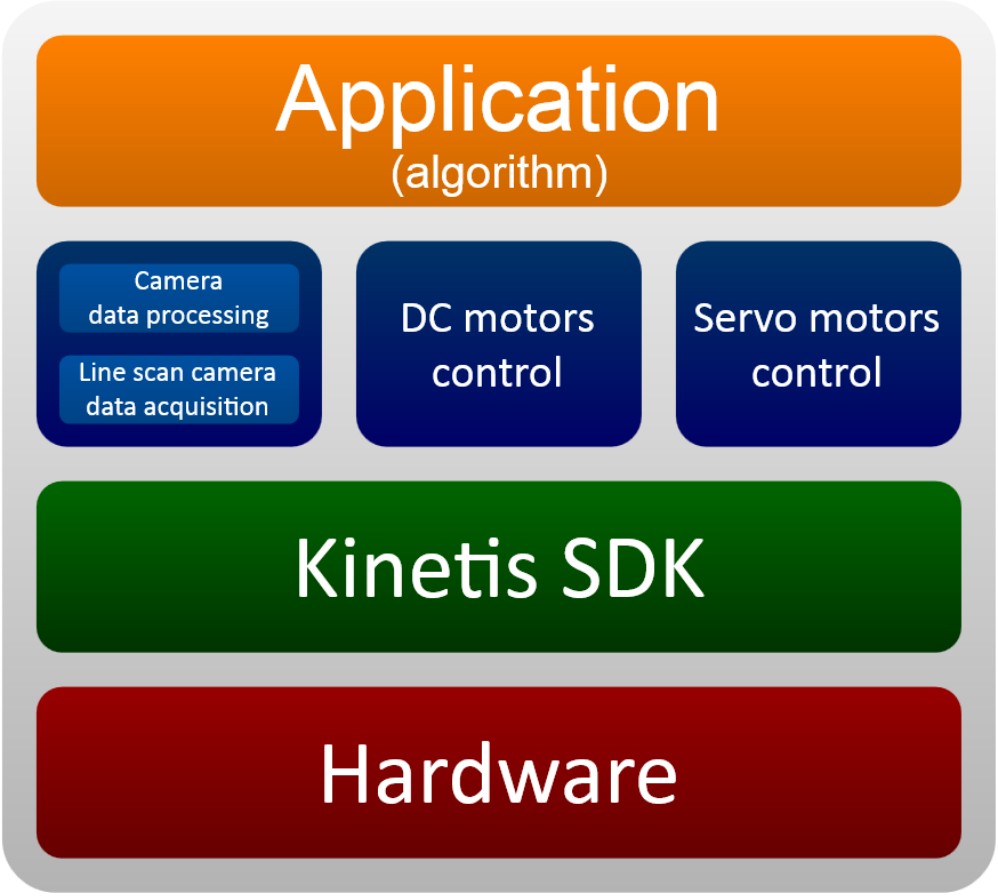


Figure 2. The software architecture that describes this example.

2. An ideal capture from an ideal track

The image below shows an ideal track. This has a background purely white with a black line exactly in the center. This image helps to show a simple way to detect where it is located, the reference frame depends on the camera position in the car.

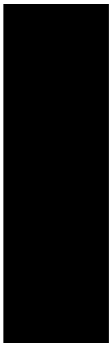


Figure 3. An ideal snapshot of a track.

The image below shows the respective data that should be acquired by the linear camera. The white color is detected as a full voltage while the black color is detected as zero voltage.

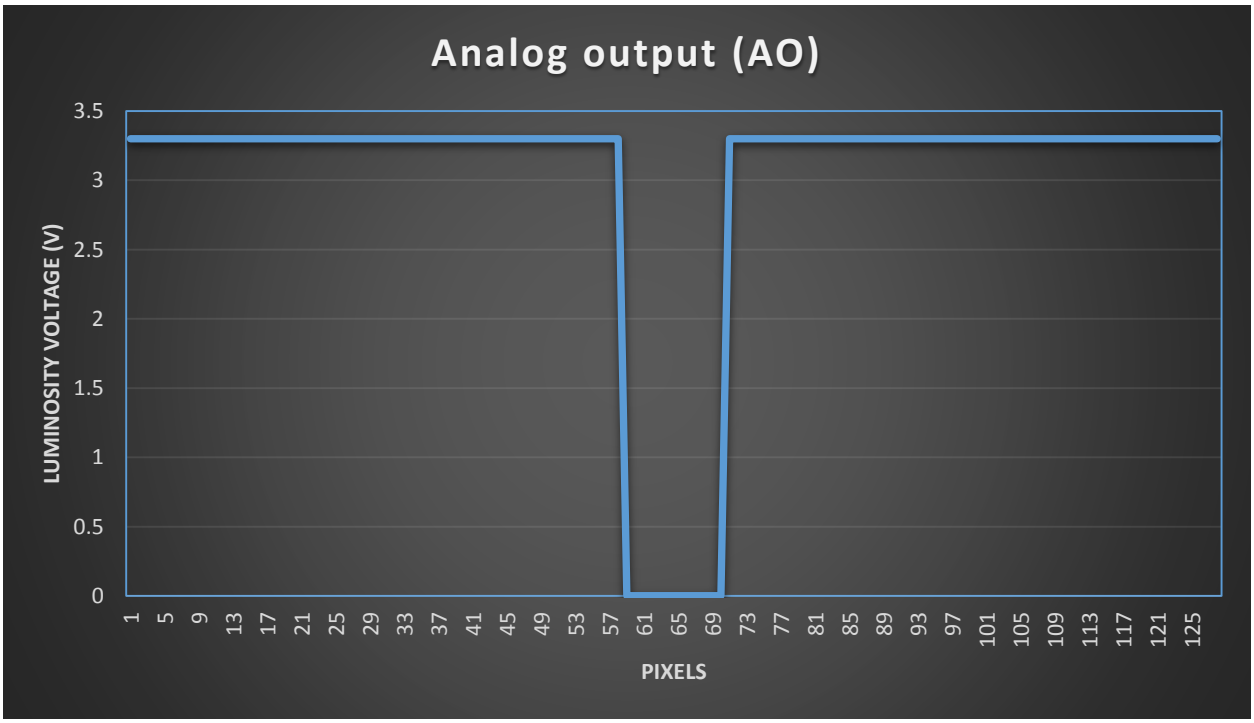


Figure 4. The acquired data from an ideal track.

The image shown below is the image taken from the camera and its central derivative, it can be seen that the maximum and the minimum of the derivative are the respective edges of the black line so it is very easy to detect the track's center, which is the average of the indexes of the minimum and the maximum values in the central derivative.

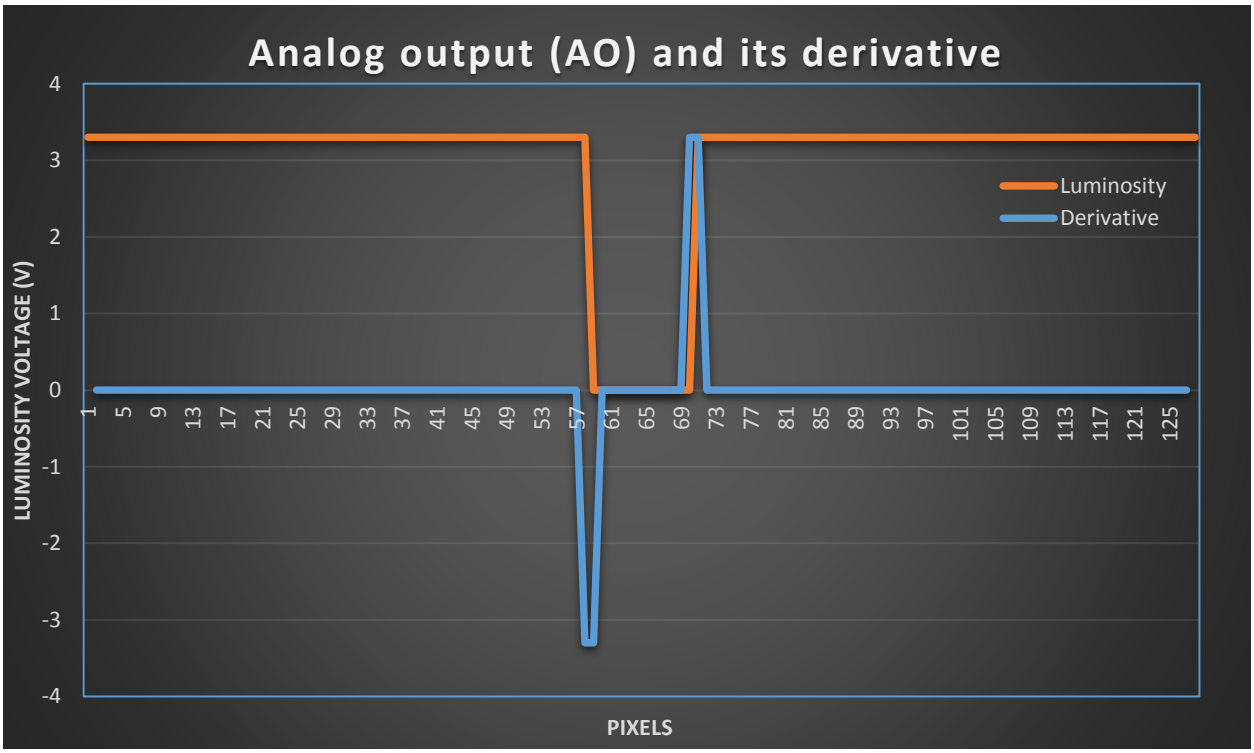


Figure 5. The raw data acquired from an ideal track and its central derivative.

3. Detecting the track's center

In this example, it was built a function to compute the track's center from an array of the raw data taken from the camera. Since this demo supports up to two cameras and it is double buffered, this function needs that the camera and the buffer to be processed are specified as parameters.

The returned value is the computed center of the track in pixels, in this case from 0 to 127.

```
uint8_t TFC_LineDetection_ComputeTrackCenter(tfc_shield_linear_camera_t
cameraToProcess, tfc_shield_linear_camera_buffers_t bufferToProcess)
{
	/* Counter to process the camera's buffer. */
	uint8_t TFC_LineDetection_PixelToProcess;

	/* Variable to store the value of the derivative. */
	int32_t TFC_LineDetection_TemporalDerivative;

	/* Variables to compute the maximum and the minimum of the line's derivative*/
	//Variable initialized with the minimum value.
	int32_t TFC_LineDetection_MaximumDerivative = 0x80000000;
	//Variable initialized with the maximum value.
	int32_t TFC_LineDetection_MinimumDerivative = 0x7FFFFFFF;

	/* Indexes of the maximum and the minimum of the line's derivative. */
	int32_t TFC_LineDetection_MaximumIndex = 0;
	int32_t TFC_LineDetection_MinimumIndex = 0;

	/* Pointer to the camera's buffer to process. */
	uint16_t* TFC_LineDetection_BufferToCompute =
TFC_LinearCamera[cameraToProcess][bufferToProcess];

	/* Process the full array. */
	for(TFC_LineDetection_PixelToProcess = 0; TFC_LineDetection_PixelToProcess <
TFC_LINEAR_CAMERA_PIXELS - 2; TFC_LineDetection_PixelToProcess++)
	{
		/* Compute the central derivative (multiplied by two). */
		TFC_LineDetection_TemporalDerivative =
TFC_LineDetection_BufferToCompute[TFC_LineDetection_PixelToProcess + 2] -
TFC_LineDetection_BufferToCompute[TFC_LineDetection_PixelToProcess];

		/* Look for the maximum derivative. */
		if(TFC_LineDetection_TemporalDerivative >
TFC_LineDetection_MaximumDerivative)
		{
			/* Update the maximum value. */
			TFC_LineDetection_MaximumDerivative =
TFC_LineDetection_TemporalDerivative;
			/* The actual pixel is a maximum. */
```

```
                    TFC_LineDetection_MaximumIndex =
TFC_LineDetection_PixelToProcess;
             }

             /* Look for the minimum derivative. */
             if(TFC_LineDetection_TemporalDerivative <
TFC_LineDetection_MinimumDerivative)
             {
                    /* Update the minimum value. */
                    TFC_LineDetection_MinimumDerivative =
TFC_LineDetection_TemporalDerivative;
                    /* The actual pixel is a maximum. */
                    TFC_LineDetection_MinimumIndex =
TFC_LineDetection_PixelToProcess;
             }
      }

/* The track's center is in the average of the maximum and the minimum's indexes. */
      return (TFC_LineDetection_MaximumIndex + TFC_LineDetection_MinimumIndex) / 2;
}
```

4. Results

An application to demonstrate the ease of use of the new APIs for the car's movement and the line detection was built. This application uses this algorithm and prints the computed center in the console. Also, a simple proportional control is used to drive the steering servomotor.

```
/* FRDM-TFC initialization. */
TFC_Shield_Init();

/* Wait until the SW button is pressed. This will prevent that the motors start when
the board is initially powered on. */
while(!TFC_SHIELD_SW1_READ)
{
	/* Blink the four TFC-SHIELD LEDs to show that the smart car is waiting for a
push in the SW1. */
	static uint16_t counter = 0;
	if(!--counter)
	{
		TFC_SHIELD_LED1_TOGGLE;
		TFC_SHIELD_LED2_TOGGLE;
		TFC_SHIELD_LED3_TOGGLE;
		TFC_SHIELD_LED4_TOGGLE;
	}
}

/* Enable the H-bridge for the motors. */
TFC_MOTORS_ENABLE_ON;

for (;;)
{
	/* If a new line has been acquired. */
	if(TFC_LinearCamera_ReadFinished)
	{
		TFC_LinearCamera_ReadFinished = 0;

		/* Print the information about the lines. */
		printf("Track center camera 1: %d\n\r",
	TFC_LinearCamera_TrackCenter[kTFCCamera1][TFC_LinearCamera_BufferToUse]);
		printf("Track center camera 2: %d\n\n\r",
	TFC_LinearCamera_TrackCenter[kTFCCamera2][TFC_LinearCamera_BufferToUse]);
	}

	/* Update the speed and the steering. THe POT1 controls the CD motors and the
POT2 the servomotors. */
	TFC_Motors_SetSpeedA(TFC_Shield_ADC_ReadValues[kTFCChnPot1] / 327 - 100);
	TFC_Motors_SetSpeedB(TFC_Shield_ADC_ReadValues[kTFCChnPot1] / 327 - 100);
	TFC_Motors_SetPositionSteering(TFC_LinearCamera_TrackCenter[kTFCCamera1][TFC_L
inearCamera_BufferToUse] - TFC_LINEAR_CAMERA_PIXELS / 2);
	TFC_Motors_SetPositionSpare(TFC_Shield_ADC_ReadValues[kTFCChnPot2] / 327-100);
```

```
    /* Enable and disable the motors. */
    if(TFC_SHIELD_SW1_READ)
    {
        /* Enable the H-bridge for the motors. */
        TFC_MOTORS_ENABLE_ON;
    }

    if(TFC_SHIELD_SW2_READ)
    {
        /* Disable the H-bridge for the motors. */
        TFC_MOTORS_ENABLE_OFF;
    }

    /* Dummy code to use the GPIOs. */
    if(TFC_SHIELD_DIP1_READ)
    {
        TFC_SHIELD_LED1_ON;
    }
    else
    {
        TFC_SHIELD_LED1_OFF;
    }

    if(TFC_SHIELD_DIP2_READ)
    {
        TFC_SHIELD_LED2_ON;
    }
    else
    {
        TFC_SHIELD_LED2_OFF;
    }

    if(TFC_SHIELD_DIP3_READ)
    {
        TFC_SHIELD_LED3_ON;
    }
    else
    {
        TFC_SHIELD_LED3_OFF;
    }

    if(TFC_SHIELD_DIP4_READ)
    {
        TFC_SHIELD_LED4_ON;
    }
    else
    {
        TFC_SHIELD_LED4_OFF;
    }
}
```

5. Conclusions

This document has demonstrated how easy the line detection in the track of the Freescale Cup Smart Car Race can be. This algorithm can be as robust as the programmer wants and also another track can be used just by changing the function in the TFC_LineDetection.c file. This project is a good starting point to create an application with the Freescale Cup smart car.