TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department of Electrical Power Engineering and Mechatronics

GROUND SPEED SENSOR BASED ON TEMPLATE MATCHING

MALLI SOBITAMISE PÕHIMÕTTEL TÖÖTAV MAAPINNA KIIRUSE ANDUR

BACHELOR THESIS

Student: Henri Johann Norden

Student code: 178945EAAB

Supervisor: Martin Jaanus, Senior Lecturer

Tallinn, 2020

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.

No academic degree has been applied for based on this material.

All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." 2020

Author: *signed digitally*

/ signature /

Thesis is in accordance with terms and requirements

"....." 2020

Supervisor: *signed digitally*

/ signature/

Accepted for defense

"....."..... 2020

Chairman of theses defense commission: *signed digitally*

/ name and signature /

Abstract

Author: Henri Johann Norden *Type of the work:* Bachelor Thesis

Title: Ground speed sensor based on template matching

Date: 2020-05-20 64 *pages*

University: Tallinn University of Technology

School: School of Engineering

Department: Department of Electrical Power Engineering and Mechatronics

Supervisor(s) of the thesis: Senior Lecturer Martin Jaanus

Abstract:

The purpose of this thesis is to provide an alternative approach to currently available non-contact Ground Speed Sensors for high-performance automotive applications. Template matching allows for the utilization of off-the-shelf machine vision components, simplifying sensor design and facilitating the configuration of a sensor specifically for the required performance or cost. A working template matching GSS is developed and the important aspects of the development process are described.

A custom lighting solution is developed to eliminate all effects of natural sunlight/shadows on the camera and to generate short intense pulses of light, allowing the camera to achieve shorter exposures than otherwise permitted by the camera's hardware. Additionally, an optimized software solution is developed. An efficient multi-threaded algorithm is designed to maximize utilization of the embedded computer's GPU. An alternative approach to the two-step low-resolution→high-resolution template matching optimization technique is described, considerably limiting the area to be processed by template matching with only a single template matching run.

Keywords: ground speed sensor, embedded automotive machine vision, template matching

Lühikokkuvõte

Autor: Henri Johann Norden *Lõputöö liik:* Bakalaureusetöö

Töö pealkiri: Malli sobitamise põhimõttel töötav maapinna kiiruse andur

Kuupäev: 2020-05-20 64 *lk*

Ülikool: Tallinna Tehnikaülikool

Teaduskond: Inseneriteaduskond

Instituut: Elektroenergeetika ja mehhatroonika instituut

Töö juhendaja(d): vanemlektor Martin Jaanus

Sisu kirjeldus:

Antud töö eesmärk on esitada alternatiivne lähenemine olemasolevatele autotööstuses kasutamiseks mõeldud kontaktivabadele maapinna kiiruse anduritele. Malli sobitamine võimaldab kasutada olemasolevaid masinnägemiskomponente, seeläbi lihtsustades anduri enda ehitust ja võimaldades konfigureerida andurit saavutama täpselt vajaminevat jõudlust või hinda. Antud töös arendatakse töötav maapinna kiiruse andur ja kirjeldatakse arendusprotsessi olulisi aspekte.

Luuakse valgustussüsteem, mis kõrvaldab loomuliku päikesevalguse/varjude mõju kaamerale ja tekitab lühikesi intensiivseid valgusimpulsse, võimaldades seeläbi kaameral saavutada kaamera riistvara piiridest lühemaid säriaegu. Lisaks arendatakse optimeeritud tarkvarakomplekt. Disainitakse efektiivne paralleelselt töötav algoritm, mis maksimeerib sardsüsteemi graafikaprotsessori kasutust. Kirjeldatakse kahesammulise madala resolutsiooniga→kõrge resolutsiooniga malli sobitamise optimeerimismeetodi asemele sobivat ühesammulist ja malli sobitamisega töödeldavat ala oluliselt vähendavat meetodit.

Märksõnad: maapinna kiiruse andur, masinnägemine, autotööstus, sardsüsteem, malli sobitamine

Table of contents

Preface

This thesis was completed in close cooperation with Formula Student Team Tallinn (registered as MTÜ Tudengi Vormel in Estonia), who provided the topic for the thesis. The sensor system described in this thesis was deployed on the team's 2019 formula car, FEST19. The author would like to thank the team for the opportunity and all the provided support.

List of abbreviations and symbols

CAN	Controller Area Network
CMOS	Complementary metal–oxide–semiconductor
CPU	Central Processing Unit
CUDA	A proprietary platform & API for general purpose parallel computing using Nvidia GPUs, developed by the Nvidia Corporation.
f-number	Ratio of an optical system's (e.g. lens') focal length to the diameter of the entrance pupil (for a camera lens, the entrance pupil is the virtual image of the physical aperture as seen through the front of the lens). [1, 2] Also known as the focal ratio, f-ratio or f-stop. [1]
FOV	Field-of-view
fps	Frames per second When used as a unit, x fps is equivalent to a framerate of x Hz.
FWHM	Full Width at Half Maximum
GPU	Graphics Processing Unit
GSS	Ground Speed Sensor
LED	Light Emitting Diode
MOSFET	Metal–Oxide–Semiconductor Field-Effect Transistor or metal–oxide–silicon transistor
NMOS	n-channel MOSFET
px	Pixel
QE	Quantum Efficiency – incident photon to converted electron (IPCE) ratio. [3]

Introduction

Precise speed measurements are required within the automotive industry for both retrospective vehicle dynamics analysis and active real-time stability assist systems. The approach common in consumer cars of estimating the vehicle's speed from a wheel's rotational speed is inadequate due to difficult-to-measure factors, such as tire slip and tire deformation. Therefore, an external sensor is required for accurate measurements. Various commercial solutions exist for this very purpose; these can be categorized into contact and non-contact/contactless sensors.

Contact sensors, such as the "fifth wheel", employ an additional wheel and estimate the vehicle's speed based on the extra wheel's rotational speed. By using an additional, non-driven, non-load-bearing wheel, the parasitic factors mentioned earlier are reduced, but not eliminated. Non-contact sensors use various techniques to measure a vehicle's actual speed without any friction-dependent component (more details in chapter 1). They can also provide more advanced functionality, such as distinct longitudinal and lateral velocity measurements (a "2-axis" GSS). Non-contact sensors are therefore used to analyse cornering scenarios and are suitable for use in high-performance stability assist systems.

The purpose of this thesis is to provide a novel configurable approach to the non-contact GSS, utilizing a high-performance system-on-module computer for template matching (chapter 2). The process of developing a functional prototype is described along with the choice of hardware components based on desired sensor specifications (chapter 3). As the sensor's main parts, the camera and lens, are relatively common off-the-shelf machine vision components, it is possible to design a range of ground speed sensors with different cost and operating parameters just by changing these parts. Consequently, as higher resolution camera models are released, the resolution and accuracy of the proposed GSS can also be increased.

In addition to off-the-shelf components, a custom low-power and low-cost lighting solution is developed to vastly improve the effective exposure time of any global shutter monochrome camera (3.5). The described solution could also be used in other applications where an exposure time lower than the range supported by a camera is required. This requirement could be, for example, due to a need to capture a sharp image of a fast-moving object, such as is the case with the proposed GSS. During testing, sharp sub-microsecond exposures with a good dynamic range were achieved using a CMOS sensor with a 54 μs minimal exposure time.

The software for the GSS is written in C++ and implements CAN communication for compatibility with automotive applications (chapter 4). The OpenCV open source computer vision library is used for its CUDA implementation of various template matching methods. Several noteworthy

optimizations are made to improve the measurement frequency of the sensor. A multithreaded algorithm is developed to ensure maximal utilization of the GPU, which runs the template matching algorithm (4.4). A common method for reducing processing time without sacrificing precision is applying template matching first on a reduced-resolution frame and subsequently on a limited region on the full-resolution frame. By considering the inertia of the measured physical system, an alternative algorithm is developed, which skips the reduced-resolution template matching and instead selects the limited region based on the previous template matching result (4.7).

1. Analysis of available commercial non-contact sensors

Currently, the two most common concepts for non-contact ground speed measurement are GNSS and optical, according to VBOX automotive [4]. A comparison of one high-performance GSS of each type is given in Table 1.1. Additionally, a radar-based sensor is included, however the expected use case for the sensor is automated agriculture instead of analysis of high-performance vehicle dynamics and this is also clear from its significantly lower specifications. Radar-based systems seem to be more suited for applications where the highest possible precision is not necessary and notably, the listed radar-based GSS does offer a better price-performance ratio than the high-performance sensors.

The high price is the main disadvantage of the GNSS and optical sensors. With the possibility of using more generic machine-vision components, such as a regular camera, lens and embedded computer, in a template matching based GSS might provide an alternative of a sensor with adequate performance for a price-point between the low-end (radar) and high-end sensor systems. Such a prospect will be explored within this thesis. A template matching based GSS might also be able to compete directly with the high-end sensors, with minimal additional development effort, as upgrading the sensors components would result directly in improved performance.

Figure 1.1. VBOX RLVBSS series [5]

Figure 1.2. Kistler Correvit SFII [6]

Table 1.1. Comparison of the developed machine vision GSS system and available commercial sensors

	Developed machine vision GSS system	**DICKEY-john Radar II [7]**	**VBOX RLVBSS100 [8, 5]**	**Kistler Correvit SFII [6, 9]**
Type	Machine vision	Doppler radar	GNSS	Optical
Cost	~ 1500 €[1]	$\$798 \approx 730$ € [10, 11]	554548 INR ≈ 7550 € [12, 13]	$> 10\,000$ € [14]
Weight	500 g	2.05 kg	250 g	670 g
Power	< 18 W	< 9.6 W	3.7 W	21 W
Working distance	150 mm (adjustable in design)	(61 … 240) cm *Mounted at* 35°	- (any)	(180 ± 50) mm
Signal latency	< 10 ms	< 200 ms	(7 … 10) ms	*Unknown* (moving average filter 8…512 ms)
Measurement freq.	> 100 Hz	< 70 Hz	100 Hz	250 Hz
Interface	CAN (also possible: USB, Ethernet, Wi-Fi, I2C, SPI)	Analog	CAN, RS232, digital frequency, analog	CAN, RS-232C, USB, digital frequency, analog
Min. velocity	$0.25\ {}^{\mathrm{km}}/_{\mathrm{h}}$ (equal to resolution)	$0.53\ {}^{\mathrm{km}}/_{\mathrm{h}}$	$0.1\ {}^{\mathrm{km}}/_{\mathrm{h}}$	$0.3\ {}^{\mathrm{km}}/_{\mathrm{h}}$
Max. velocity	$110\ {}^{\mathrm{km}}/_{\mathrm{h}}$ (adjustable in design)	$96.6\ {}^{\mathrm{km}}/_{\mathrm{h}}$	1000 mph $\approx 1600\ {}^{\mathrm{km}}/_{\mathrm{h}}$	$250\ {}^{\mathrm{km}}/_{\mathrm{h}}$
Resolution	$0.25\ {}^{\mathrm{km}}/_{\mathrm{h}}$ (adjustable in design)	- (analogue output only)	$0.01\ {}^{\mathrm{km}}/_{\mathrm{h}}$	$0.01\ {}^{\mathrm{m}}/_{\mathrm{s}} = 0.036\ {}^{\mathrm{km}}/_{\mathrm{h}}$
Accuracy	$\geq (0.0833\ {}^{\mathrm{km}}/_{\mathrm{h}} + 1.33\ \%)$[2]	$< 5\ \%\ (< 3.2\ {}^{\mathrm{km}}/_{\mathrm{h}})$ $< 3\%\ (> 3.2\ {}^{\mathrm{km}}/_{\mathrm{h}})$	$0.1\ {}^{\mathrm{km}}/_{\mathrm{h}}$ (4 sample average)	$< 0.5\ \%$

[1] Only includes the part cost of the system.

[2] Only a baseline is given within this thesis: see item 3.6.2.

2. Theory of operation

2.1 Template matching

Template matching is a digital image processing technique for identifying possible locations for a small image (the template) in a larger base image. There are two main approaches to template matching: feature-based and template-based. In a feature-based approach, image features, such as edges, lines, blobs and corners, are first detected and extracted from the images [15]. This results in descriptions of the template and base image that are invariant to scale, rotation, translation, illumination and blur. Then, a feature matching algorithm is run and good matches are filtered out. This is sufficient for identifying the template's location on the base image and, additionally, their homography can be found to identify the specific perspective transformation that has taken place between the two images. [16, 17]

Template-based matching is much simpler, but less efficient for high-resolution images and intolerant of perspective transformations between template and base images. In template-based matching, the template is slid over the base image one pixel at a time, left to right and top to bottom. At each location a metric representing the similarity of the template to that specific are of the base image is calculated. The output is therefore a 2-dimensional array of probability values, where the size of the array is equal to the base image's size. The minimum/maximum value (depending on the used metric) is found and this point is the single best match for the template on the given base image. A visual representation of the array with a blue→green color scheme is given in Figure 2.1, with the best matching location encircled in red. [18]

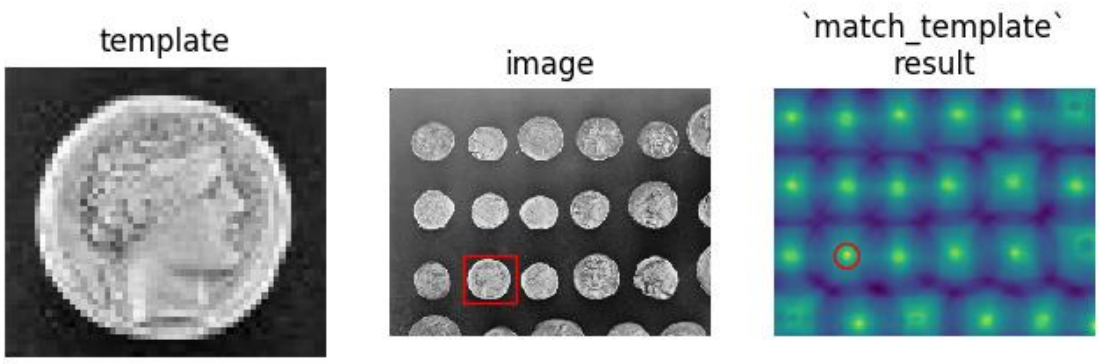


Figure 2.1. Template-based matching [19]

2.2 Ground speed measurement with template matching

If a camera is pointed perpendicular to the ground, template matching can be used to measure the speed at which the ground moves past the camera. This can be accomplished with the following general algorithm:

1. The camera captures 2 frames with a shift of t seconds between the frames. The value of t is determined by the framerate, f, of the camera:

$$t\ [\mathrm{s}] = {}^{1}/_{f\ [\mathrm{fps}]} \tag{2.1}$$

   The value of t must be low enough (or, the framerate must be high enough) that a chosen area of the first frame (the template) is also captured in its entirety on the second frame.

2. Template matching is run on the 2nd frame, using the chosen area from the 1st frame as the template. The difference between the resulting best match location for the template on the 2nd frame and the original location for the template on the 1st frame is calculated. The physical movement of the chosen area during the last t seconds is now known: x pixels for the longitudinal axis and y pixels for the lateral axis.

3. The physical size of a pixel must be determined to determine the physical speed for the pixel shift values x and y. The methods for this are described in section 3.4.

4. Finally, the ground velocity can be calculated:

$$v_x = \frac{x \cdot px_{size}}{t}, \qquad v_y = \frac{y \cdot px_{size}}{t}, \tag{2.2, 2.3}$$

   v_x, v_y – longitudinal and lateral velocities, $[{}^{\mathrm{m}}/_{\mathrm{s}}]$,
   x, y – longitudinal and lateral shift in a chosen area between 2 captured frames, $[\mathrm{px}]$,
   px_{size} – physical size of a pixel, $[{}^{\mathrm{m}}/_{\mathrm{px}}]$,
   t – time shift between the 2 captured frames, $[\mathrm{s}]$.

5. Optional filtering can be applied to the result. A three-value median filter, as used in the proposed GSS, allows the system to tolerate the effect of a single random mismatch. As a downside, a filter introduces additional signal latency.

The following figure shows an example of high longitudinal velocity (x-axis) and negligible lateral velocity (y-axis) with both captured frames side-by-side. The area chosen for the template is within the dark-gray rectangle and the template matching result is within the white rectangle.

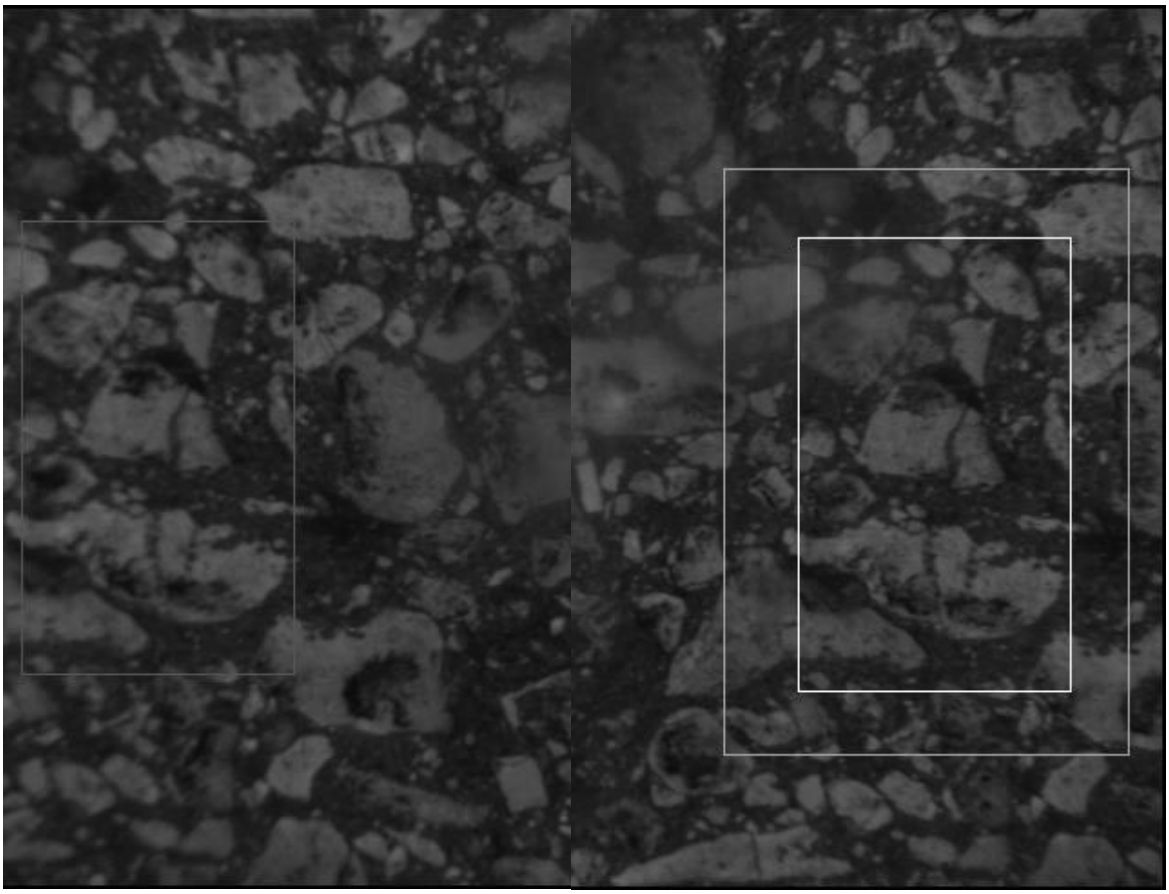


Figure 2.2. Template matching in the GSS (*left*: left half of 1st frame; *right*: right half of 2nd frame)

The above images are of asphalt at a kart racing track. The images show a good rock pattern exists for template matching to work. The patterns are, however, very random with few large defining features. It therefore seems template-based matching is better suited for the task than feature-based matching. The more advanced nature of feature-based matching offering invariance to perspective transformations is also not necessary and would simply increase processing time. Although a car's yaw, roll and pitch can all cause a perspective transformation incompatible with template-based matching, the high inertia of the measured physical system and the necessarily low time between 2 captured frames ensures that an unmanageable perspective transformation cannot occur. This was later experimentally confirmed, as no mismatches attributable to perspective transformations were encountered.

2.2.1 Lighting requirements

For the lighting requirements, it is assumed that a monochrome camera is used (see point 3.3.1).

For the best matching results, each frame should have a uniform light level across the whole frame. Template-based matching compares the pixels in the template and the base image and calculates a similarity metric based on the differences in values of each pixel. Therefore, if either of the images is uniformly lighter/darker, all the pixel and output similarity metric values shift by the same amount and finding the minimum/maximum among them still results in the same match location being found. If there is a non-uniform lighting disturbance on one of the images however, only some of the pixel values are shifted and so the area on the base image with the most similar light level is likely to match the template instead of the area with the most similar texture (Figure 2.3).

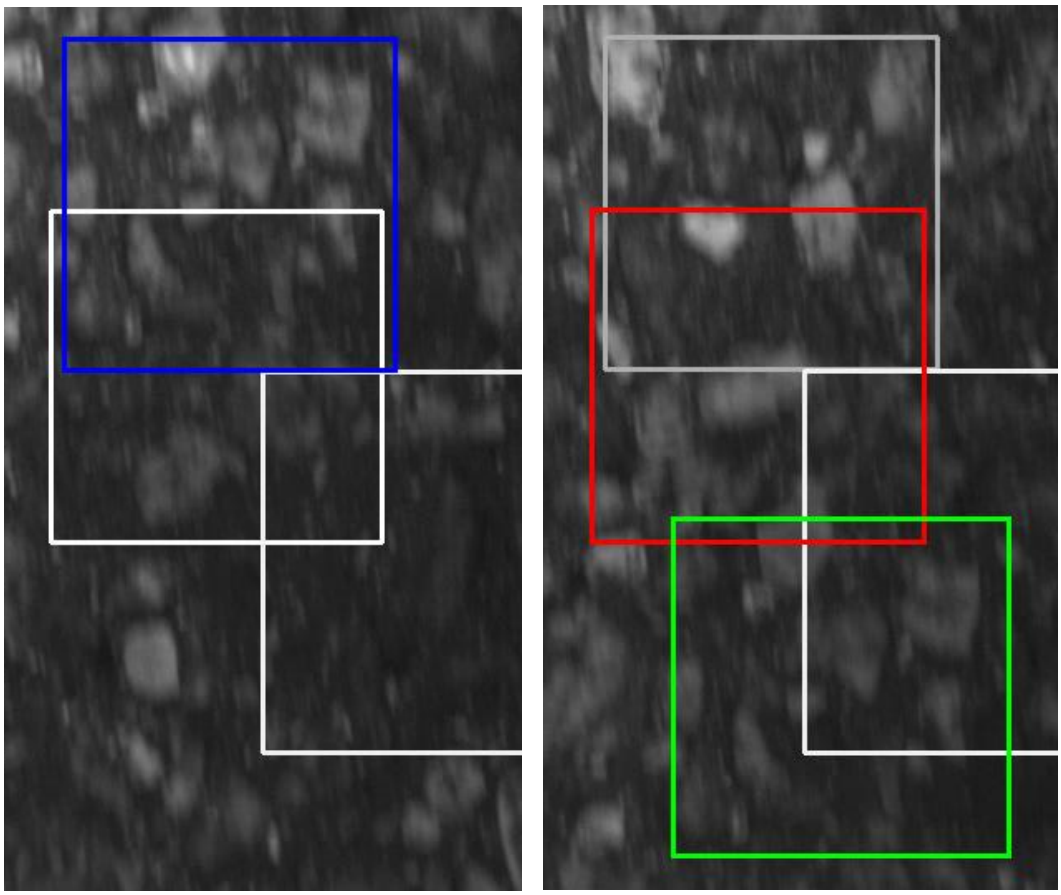


Figure 2.3. Mismatch due to lighting differences (*blue*: template, *red*: match, *green*: expected match)

Such non-uniform lighting disturbances can be easily created by uneven sunlight illumination (in Figure 2.3, the bottom part of the image is further under the car that the top) or by shadows from specific objects, such as tree branches. To mitigate the issue, the sensor system should have its own homogenous light source and should be shielded from external light. This issue is addressed in section 3.5.

3. Hardware

3.1 Specification of fundamental sensor parameters

As the proposed GSS solution could be constructed from a wide range of components, minimum requirements for the sensor output must be defined. These requirements will vary between applications; the requirements defined within this thesis are solely based on suitability for use on the Formula Student Team Tallinn FEST19 car. The established quantitative requirements (Table 3.1) were however insufficient for component selection, so additional optimization objectives were determined (Table 3.2). It should be possible to design a template matching GSS with optimal parameters for any application by following the guidelines outlined in the following sections with custom application-specific sensor requirements & objectives.

Table 3.1. Quantitative requirements

Requirement	Value	Explanation
Mounting height range (h_{cam})	$(30 \dots 300)$ mm	Within this range, parts will remain between the lowest and highest point of the monocoque. Any part must not extend below the lowest point to avoid collisions with the ground due to suspension articulation. The highest point limit was chosen to ensure mounting simplicity and aesthetic homogeneity.
Max. speed ($v_{\max}$)	$90\ {}^{\mathrm{km}}/_{\mathrm{h}}$	A velocity value never exceeded by the FEST19 in technical dynamic events (*i.e.* only exceeded in acceleration).
Measurement frequency	200 Hz	FEST19 electronic control unit control cycle frequency.

Table 3.2. Optimization objectives

Objective	Explanation
Sensor component integration simplicity	Easily integratable and interoperable components aid in reaching the goal of a working template matching GSS prototype with limited engineering hours. This objective includes mainly comparable factors, such as component documentation quality, API quality/simplicity and standard compliance, but also subjective factors, such as familiarity with and the popularity of the component's platform.
Low price	The cost of each component determines the final cost of the sensor.[1]
High GSS resolution	A higher resolution makes achieving higher accuracies possible with template matching and optional additional filtering.

[1] As participation in the Formula Student programme is voluntary and unpaid, labor costs were not directly considered during development. The first objective of using easily integratable components would however likely have an indirect lowering effect on labor costs in a commercial development scenario.

3.2 Template matching computer

Among embedded computer systems, Nvidia's Jetson series is one of the most common high-performance offerings. The Jetson TX1 is more than twice as fast than the Raspberry Pi 3 in many CPU benchmarks [20]. However, the main advantage of the Jetson series is the inclusion of a GPU with 256 CUDA cores. GPU accelerated template matching using the CUDA framework is natively supported by OpenCV, which allows to easily parallelize the demanding template matching operation and leaves the CPU free for other tasks. Image processing often naturally maps to GPU architectures and this allows for a performance increase of up to 30 times over a CPU implementation of the same algorithm [21].

The Jetson TX1 module was chosen, because it offers all the benefits of the Jetson series and a spare module was available to the Formula Student team. It is noteworthy that newer modules offering better performance/price are available at the time of writing: TX2, Xavier NX, AGX Xavier and Nano [22]. The Jetson TX1 is a System-on-Chip module, requiring a carrier board to take advantage of all or some of the features of the module. The development board offered by Nvidia has a large footprint, so a smaller 3rd party carrier was opted for: Auvidea's J120. The (50x110) mm J120 notably offers 2 USB 3.0 ports, Gigabit Ethernet, mini-HDMI, a CAN controller, SPI, I^2C, UART and a PCIe M.2 SSD slot [23].

Figure 3.1. Jetson TX2 (identical dimensions to TX1) mounted on an Auvidea J120 [24]

3.3 Camera & lens

The choice of the camera and its lens is a complicated problem, as the GSS's main measurement characteristics are determined by a combination of the camera's and lens' parameters and the camera's mounting height. Some notable effects of changing these parameters are listed in the following table, with each row representing an input parameter and each column a change potentially resulting from modifying an input parameter.

Table 3.3. Effects of changing camera/lens parameters ('+' denotes a positive and '-' a negative correlation)

		GSS system output parameters					**Cost**		**Image**			**Other**
Input (chosen) parameters		**Min speed decrease[1]**	**Max speed**	**Resolution**	**Frequency**	**Signal latency decrease[4]**	**Camera/lens price**	**Required processing power**	**Blur decrease**	**Required light decrease[4]**	**Angular FOV decrease[4,2]**	**Min mounting height decrease[4]**
Camera	**Framerate**	+		+	+	+	+	+				+
	Resolution	+		+			+	+				
	Exposure time decrease[4]		+	+			+		+	-		
	Sensor size						+			+	-	
Lens	**Focal length**	+	-	+					-		+	-
	f-number								+	-		
Other	**Mounting height decrease[4]**	+	-	+							+	

[1] A decrease in the parameter is specified for parameters where a lower value desirable. This way, a positive correlation (+) in the resulting table most often reflects a 'desirable' relationship and a negative correlation (-) reflects an 'undesirable' one.

[2] A decrease in angular FOV reduces the volume of free space which must be reserved for the camera, so that no objects come into the camera's view.

The main parameters considered were camera framerate, camera resolution, camera mounting height, lens focal length and additionally the prices of the respective components. The following figures exemplify the changes to the final system caused by changes in these parameters. The formulas for quantitive analysis of these effects are given within the following items of this section.

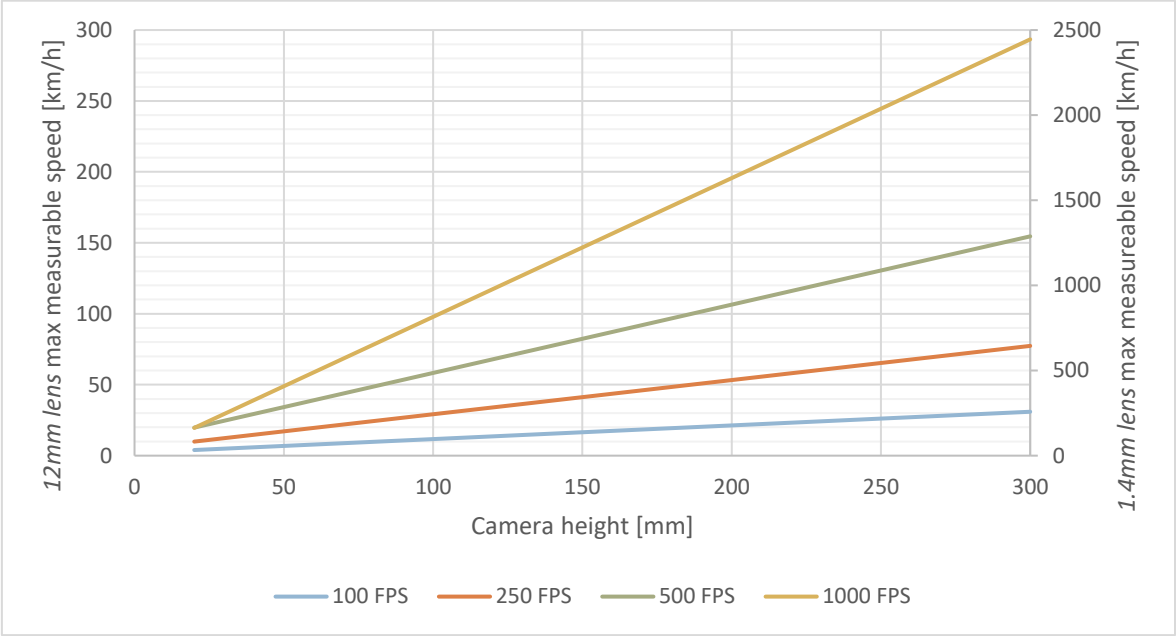


Figure 3.2. The effect of camera framerate, mount height and lens focal length on max. measurable speed

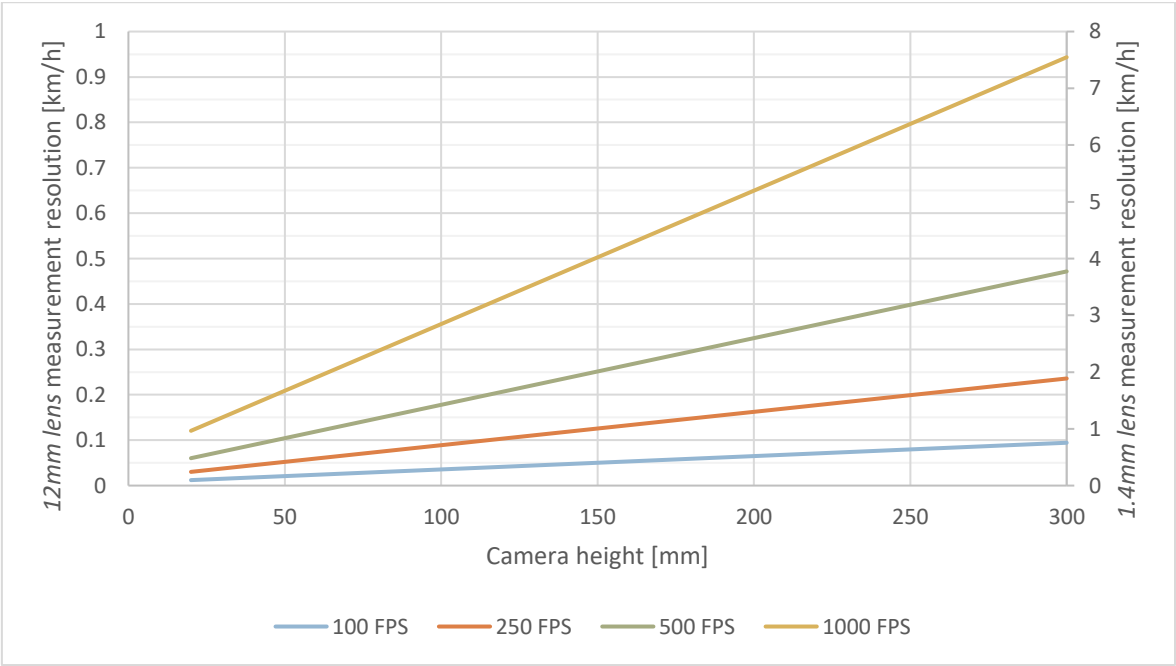


Figure 3.3. The effect of camera framerate, mount height and lens focal length on measurement resolution

3.3.1 Camera

Figure 3.4. Ximea xiQ MQ003MG-CM [23]

The Ximea xiQ camera series was found to be best suited for the application. Ximea officially supports the Jetson TX1 and has well-documented API packages available for C++, .NET and Python. The xiQ MQ003MG-CM camera has a low cost, small dimensions of (26 x 26 x 24) mm, low mass of 26 grams, low power consumption of 1.5 W, USB 3 connectivity and a CMOSIS CMV300 1/3" monochrome global shutter sensor. Monochrome sensors are beneficial as they are more sensitive due to the lack of color filters and asphalt possesses very little color information regardless. The camera is capable of capturing (648x488) pixel frames in 54µs exposures at 500 fps, whereas most similarly priced higher resolution cameras must be limited to a similar/lower resolution to reach such a framerate. Additionally, the camera supports C/CS mount lens and has a GPIO port. It must be noted that it is possible for newer superior camera models to exist at the time of writing. [25]

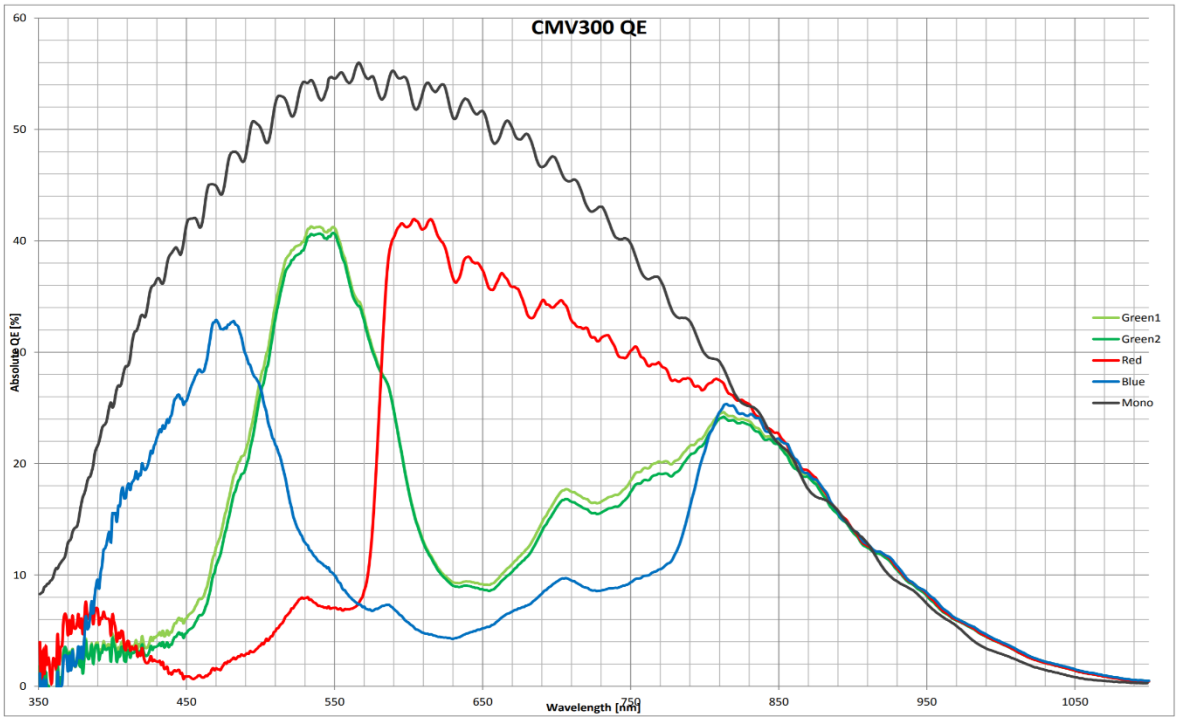


Figure 3.5. Quantum efficiency of the CMOSIS CMV300 sensor used by the MQ003MG-CM camera [26]

3.3.2 Parameter calculations – mounting height & lens focal length

Given the choice of camera, the following constants were fixed to determine the mounting height for the camera lens, h_{cam}, and the desired lens focal length, fl:

Frame width	$w = 648$ px,
Frame height	$h = 488$ px,
Sensor width	$w_{sensor} = 4.8$ mm,
Sensor height	$h_{sensor} = 3.6$ mm,
Reserved width for reversing (negative velocity)	$w_{reverse} = 10$ px,
Template width	$w_{template} = 100$ px,
Max velocity	$v_{\max} = 90\,{}^{\mathrm{km}}/_{\mathrm{h}}$,
Time between captured frames	$t_{frame} = {}^{1}/_{500\ \mathrm{fps}} = 2$ ms.

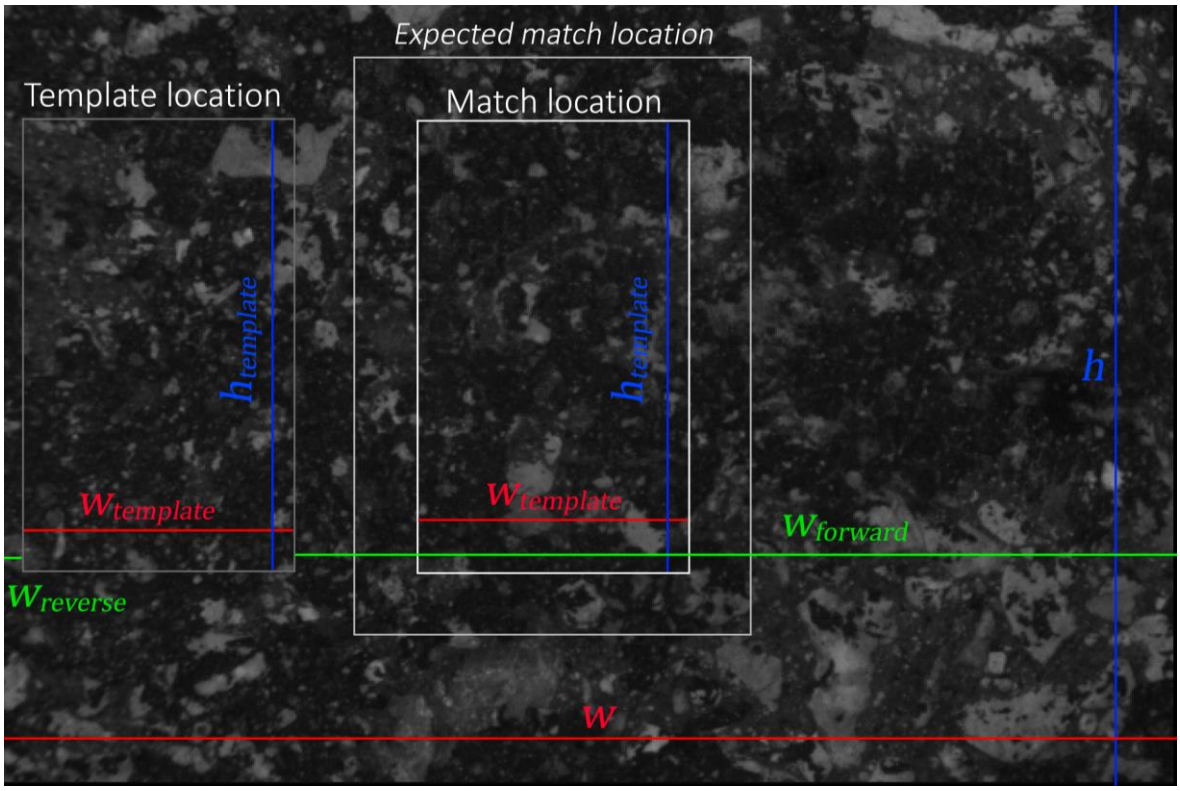


Figure 3.6. Frame dimensions

The following optical relationships exist:

$$\alpha = 2 \cdot \arctan \frac{w_{sensor}}{2 \cdot fl}, \quad (3.1)\ [27]$$

α – angle of view of the lens, [°],

fl – focal length of the lens, [mm].

$$w_{frame} = 2 \cdot \tan\left(\frac{\alpha}{2}\right) \cdot h_{cam}, \quad (3.2)\ [27]$$

w_{frame} – optical system's (camera & lens) linear field of view, [mm],

h_{cam} – camera mounting height, [mm].

The maximum and minimum measurable velocities can be found with the following formulas:

$$w_{forward} = w - w_{reverse} - w_{template} = 538 \text{ px}, \quad (3.3)$$

$w_{forward}$ – frame width usable for forward movement such that no part of the template at the match location extends beyond the frame, [px].

$$px_{size} = \frac{w_{frame}}{w}, \quad (3.4)$$

px_{size} – the physical size of a pixel, $[{}^{\text{mm}}/_{\text{px}}]$.

$$w_{forward_{phys}} = w_{forward} \cdot px_{size}, \quad (3.5)$$

$w_{forward_{phys}}$ – usable frame width's physical size (on the ground), [mm].

$$v_{\max} = \frac{w_{forward_{phys}}}{t_{frame}}, \quad (3.6)$$

$v_{\max}$ – maximum measurable velocity, $[{}^{\text{mm}}/_{\text{ms}} = {}^{\text{m}}/_{\text{s}}]$.

$$v_{\min} = \frac{px_{size} \cdot 1 \text{ px}}{t_{frame}}, \quad (3.7)$$

$v_{\min}$ – minimum measurable velocity, which is also the resolution of the system, $[{}^{\text{mm}}/_{\text{ms}} = {}^{\text{m}}/_{\text{s}}]$.

The given formulas were implemented in an Excel spreadsheet. As the value for $v_{\max}$ is already fixed and instead the values for fl and h_{cam} are required, the Excel Solver tool was used to solve for the missing variables. With h_{cam} fixed to the extreme values of the range specified in Table 3.1, the formulas were solved for fl. The specific solution found for the acceptable range for $fl = (2.4 \dots 23.9)$ mm:

Table 3.4. Additional fixed values used to solve for focal length fl

	Max velocity $v_{\max}$	**Mounting height h_{cam}**	**Focal length fl**
Min	$90\,{}^{\mathrm{km}}/_{\mathrm{h}} = 25\,{}^{\mathrm{m}}/_{\mathrm{s}}$	30 mm	~2.4 mm
Max		300 mm	~23.9 mm

3.3.3 Lens

Within the acceptable focal length range of $(2.4 \dots 23.9)\ \mathrm{mm}$, there are additional constraints to consider. A shorter focal length results in a wider field of view with more optical distortion and requires the camera to be mounted closer to the ground (Figure 3.2, Figure 3.3; the opposite is also true for longer focal lengths). This reduces the amount of free space required but increases the system's sensitivity to changes in height. Instead, it would be ideal to remove any dependence on mounting height therefore also eliminating the effects of height changes – this could be accomplished by utilizing a telecentric lens. With a telecentric lens the magnification of an object does not depend on the lens' distance from the object. Using a telecentric lens was considered, but due to the large size of their optics and consequent high cost it was decided to use a conventional lens along with a height measurement sensor to eliminate the error.

While a short focal length results in more optical distortion and sensitivity to height variation, a long focal length results in a shallow depth of field. A shallow depth of field could mean that when the lens' height changes, the ground moves out of focus and a blurry image is created. Both optical distortion and image blur can result in mismatches during template matching, therefore it is desirable to avoid both. The Computar M1214-MP2 lens with a $12\ \mathrm{mm}$ focal length offered a middle ground between the two sides at a low cost and small physical footprint ($\varnothing 33.5\ \mathrm{mm} \times 28.2\ \mathrm{mm}$, $61.9\ \mathrm{g}$) [28].

Figure 3.7. Computar M1214-MP2 [32]

With the focal length fl fixed to $12\ \mathrm{mm}$ it is now possible to solve for the mounting height h_{cam}. With the same initial fixed parameters (given in item 3.3.2), the ideal mounting height for the lens (as measured from the ground to the outer-most lens of the lens assembly) is approximately $h_{cam} = 150\ \mathrm{mm}$. With the chosen values, the system's final maximum velocity, $v_{\max}$, can be calculated to verify the result. Using the designed Excel spreadsheet and the formulas described in the previous item, a max velocity value of $v_{\max} \approx 89.667\ {}^{\mathrm{km}}/_{\mathrm{h}}$ (close to the desired $90\ {}^{\mathrm{km}}/_{\mathrm{h}}$ max velocity) and a min velocity value of $v_{\min} \approx 0.166\ {}^{\mathrm{km}}/_{\mathrm{h}}$ is calculated.

3.3.4 Required camera exposure time

Using the previously calculated values, it can be shown that the $54\mu s$ minimum exposure time of the MQ003MG-CM on its own is inadequate for use in a high-speed automotive application such as the proposed GSS. The minimum exposure time is too high to allow a clear image to be captured while the car is in motion. The resulting motion blur negatively affects template matching performance, as contrast is lost due to pixels being assigned values averaged over their neighbours. The maximum exposure time, which avoids a pixel overlap and therefore motion blur, can be calculated with the formula:

$$t_{\exp_{req}} = \frac{px_{size} \cdot 1\ \text{px}}{v}, \tag{3.8}$$

$t_{\exp_{req}}$ – required exposure time to avoid pixel overlap at the given velocity, $[\mu s]$,

px_{size} – the physical size of a pixel, $\left[{}^{\text{mm}}/_{\text{px}}\right]$,

v – velocity, $\left[{}^{\text{mm}}/_{\mu s}\right]$.

At the chosen height of $h_{cam} = 150\ \text{mm}$ and desired maximum velocity of $v_{\max} = 90\ {}^{\text{km}}/_{\text{h}}$, an exposure time of $t_{\exp} \leq \sim 3.7\ \mu s$ is required. The $54\ \mu s$ supported minimum exposure time of the MQ003MG-CM avoids motion blur only up to a velocity of $v \approx 6.2\ {}^{\text{km}}/_{\text{h}}$. This issue is addressed in section 3.5.

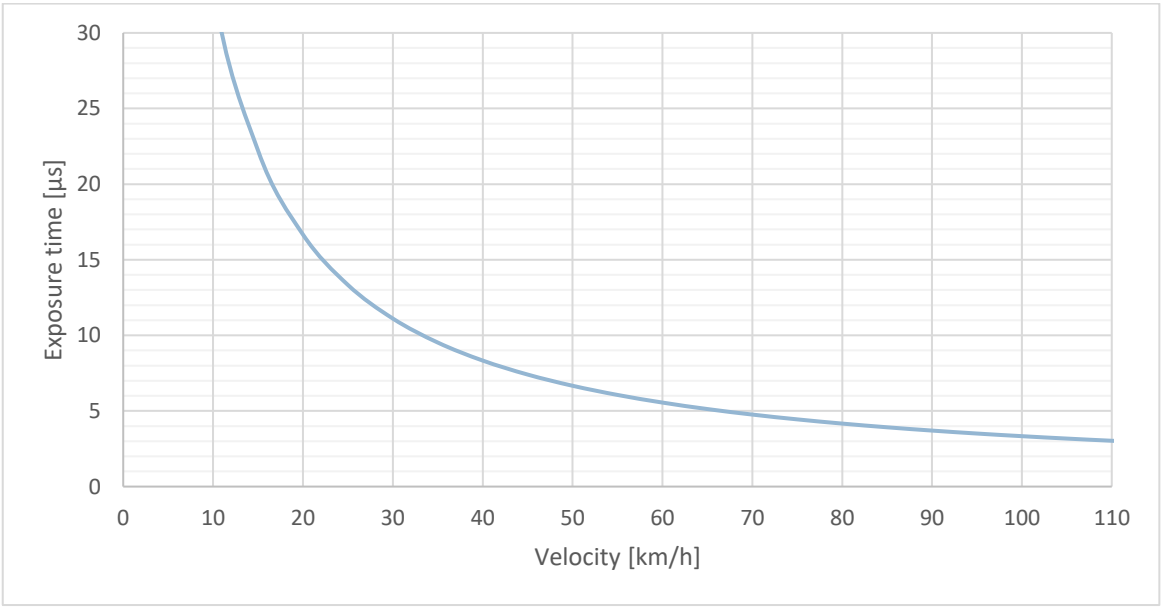


Figure 3.8. Maximum exposure time avoiding pixel overlap during an exposure

3.4 Suspension travel compensation – height sensing

Given that a telecentric lens is not used, a height reading is required to compensate for the vertical movement of the car where the GSS is mounted. The movement is chiefly caused by suspension travel and in a smaller proportion by tire deformation. It might be possible to use existing suspension position sensors to generate the car height reading, but due to the complexity of assessing momentary tire deformation, it was decided to use an external time-of-flight distance sensor instead. The low-cost STMicroelectronics' VL6180X time-of-flight proximity and ambient light sensor was chosen for this purpose. Three redundant Adafruit VL6180X development boards, each featuring a VL6810X sensor and necessary supporting components, were mounted inside the underbody of the car (~ 30 mm above ground at rest), with holes in the underbody for sensing.

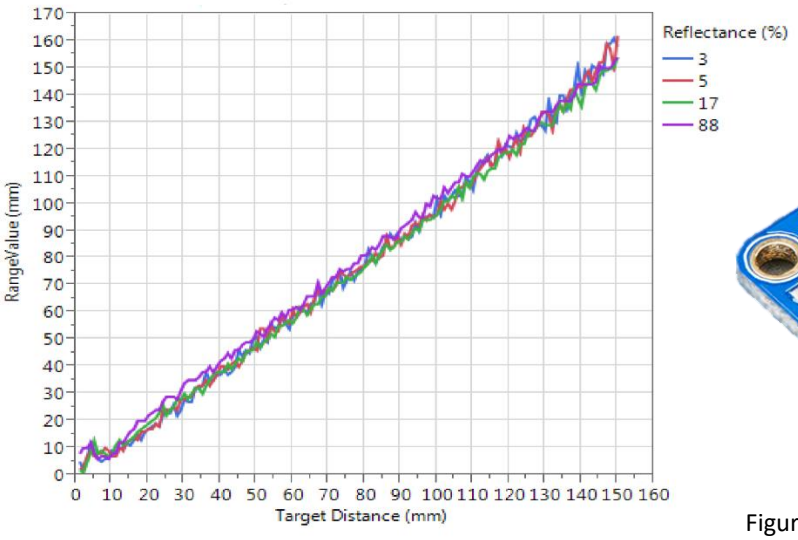


Figure 3.9. VL6180X typical ranging performance [29]

Figure 3.10. Adafruit VL6180X [33]

The VL6180X can measure with an absolute error of $err \leq 2$ mm at up to 100 mm, which is enough to cover the suspension travel range. The maximum possible error can be calculated by calculating a new max velocity value, $v'_{\max}$, with the same method as described in 3.3.3 by replacing the value of h_{cam} with $(h_{cam} + \max err) = 150 \text{ mm} + 2 \text{ mm} = 152 \text{ mm}$. This gives a new max velocity value of $v'_{\max} = 90.862 \, {}^{\text{km}}/_{\text{h}}$. The proportional error caused by the VL6180X sensor, $err_{\%}$, can now be calculated as follows:

$$err_{\%} = \pm \frac{v'_{\max} - v_{\max}}{v_{\max}} = \pm \frac{1.195}{89.667} \approx \pm 1.33 \, \% \tag{3.9}$$

During on-track testing of the proposed GSS system, the VL6180X sensor was found to be unsuitable for this application. While the maximum proportional error, $err_{\%}$, from the VL6180X is relatively large compared to commercial GSS systems, it was assumed that the possibility of mitigating this error via filtering or calibrating the proximity sensor exists. This was not the case however, as the error exhibited itself as random noise even with the sensor's averaging time set to the maximum value of $\sim 16\ \mathrm{ms}$. Calibration was also unsuccessful, as the proximity sensor's median output value could drift by as much as $1\ \mathrm{mm}$ over the course of 1 hour in a temperature and light controlled environment. Any noise from the proximity sensor directly induced noise into the GSS output, as seen in Figure 3.11.

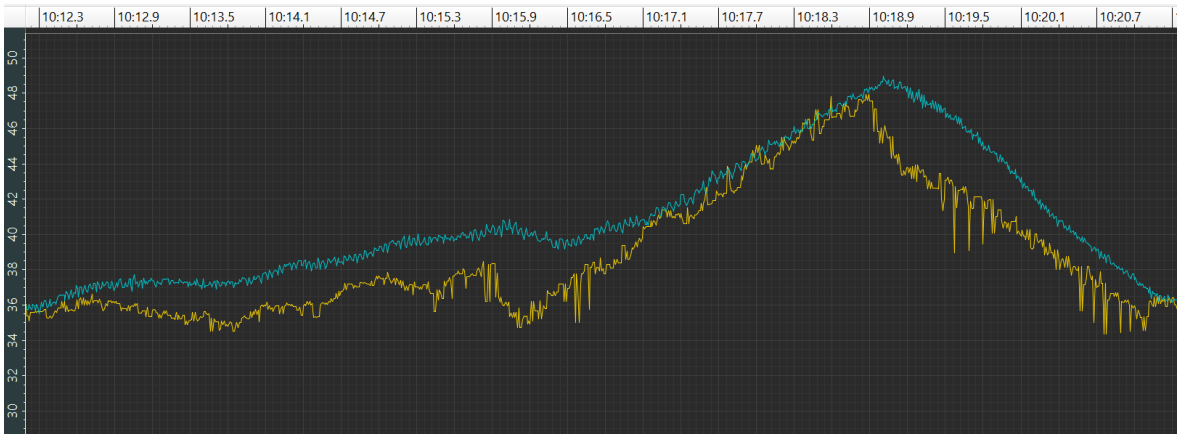


Figure 3.11. Noisy GSS linear velocity output when utilizing the VL6180X proximity sensor (yellow) compared to wheel rotational velocity sensor output (cyan)

3.5 Lighting solution

A lighting solution is required to provide adequate lighting for the low exposure times required and can additionally be used to limit the exposure time itself. To accomplish this, the camera's exposure to sunlight was limited with a $630\ \text{nm}$ band-bass filter ($(30 \pm 5)\ \text{nm}$ FWHM). With the filter, the camera's exposures are practically unaffected by natural sunlight and shadows. For lighting, high-power $621\ \text{nm}$ ($18\ \text{nm}$ FWHM) $120°$ Osram KR DMLS31.23 SMD LEDs were used (peak is at 52% QE: Figure 3.5). The LEDs are rated for a nominal current of $1\ \text{A}$ and a surge current of $2.5\ \text{A}$.

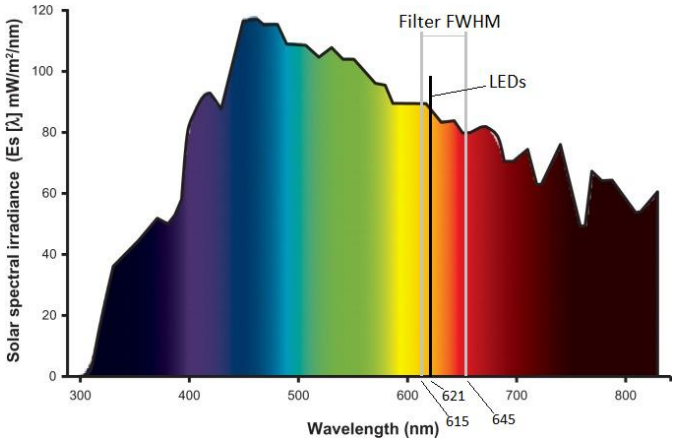


Figure 3.12. lighting band-pass filter and LED wavelengths on the solar irradiation spectrum [30]

A custom PCB was designed to interface with the camera's GPIO and generate a short intense pulse of light using LEDs. To achieve high currents a typical current-limiting resistor was omitted and current regulation was to be performed by tuning the LED driving voltage (potentiometer on A1, Figure 1.1). The worst-case operating conditions for the LEDs would be a constant 500 fps capture by the camera (no skipped frames) with 3.7 µs on-time (as per 3.3.4). At worst-case operating conditions the LEDs would be operating at a duty cycle of ${}^{3.7\ \mu\text{s}}/_{1 \div 500\ \text{fps}} = {}^{3.7\ \mu\text{s}}/_{2\ \text{ms}} = 0.00185$. As the KS DMLS31.23 datasheet abruptly cap the current capacity at $2.5\ \text{A}$ (Figure 3.13) at a pulse time > 200 times longer and a duty cycle > 2.5 times larger than desired, it was assumed the LEDs could likely be driven at currents exceeding the rating.

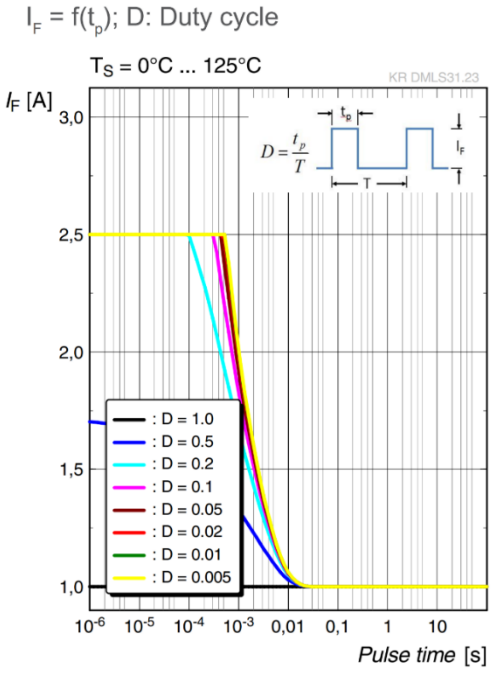


Figure 3.13. LED pulse current capacity

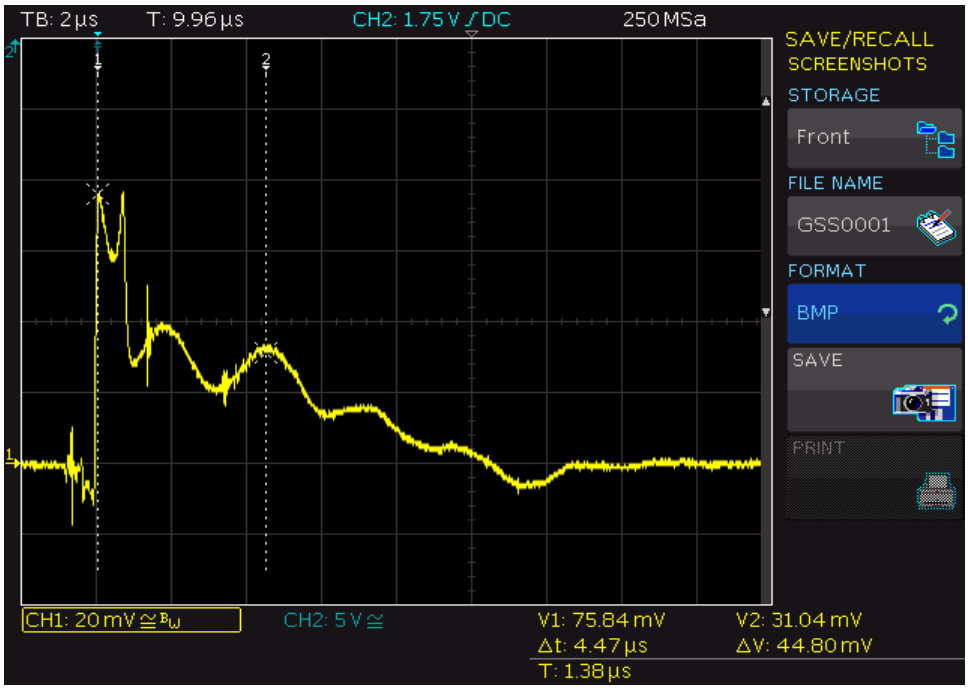


Figure 3.14. LED pulse characteristic (Y-axis scale: $^{1\ \mathrm{A}}/_{10\ \mathrm{mV}}$)

The LEDs were driven at a voltage of 6.5 V across the 2 LEDs in series (3.25 V per LED, see Figure 3.16) and peak current of 7.6 A for 1 µs pulses. The LED driving waveform, as seen in Figure 3.14, indicates parasitic inductance in the circuit and poor LED trigger NMOS driving performance. The poor NMOS driving performance could be attributed to a footprint design fault, which forced the omission of the high-current push-pull MOSFET driver (DRV1, Figure 3.15). The circuit was completed by having the single 555 timer drive 20 LED MOSFETs directly.

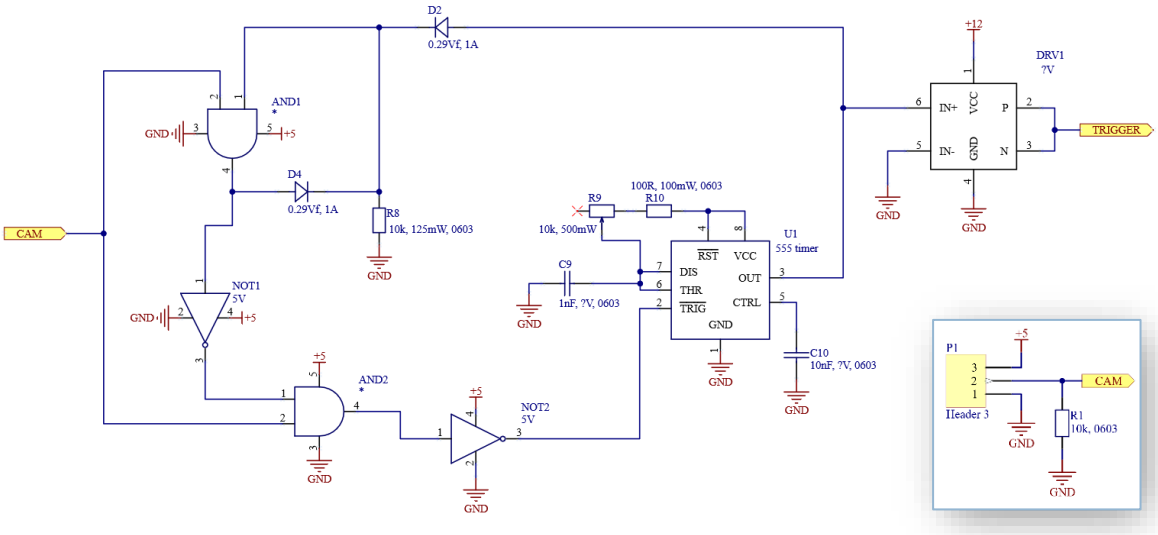


Figure 3.15. LED trigger signal generation circuit

The LED performance degraded during testing, but it was likely caused by an EMF noise issue instead of driving the LEDs over their rated values. Due to noise on the trigger signal cable generated by the physical proximity to the formula car's battery's HV wires, on initial in-car tests the LEDs were triggered randomly and much more often than the camera's maximum framerate of 500 Hz. The trigger signal noise was eliminated by replacing the unshielded signal cable with a shielded one, but only when the shielding was connected to both the camera's shield ground and the PCB's ground plane. The incident had likely irreversibly damaged the LEDs, since the PCB's voltage and time parameters had been first set and the brightness assessed using a bench-top power supply – the LEDs were dimmer and the camera's gain had to be increased from 0 to 2 dB to compensate.

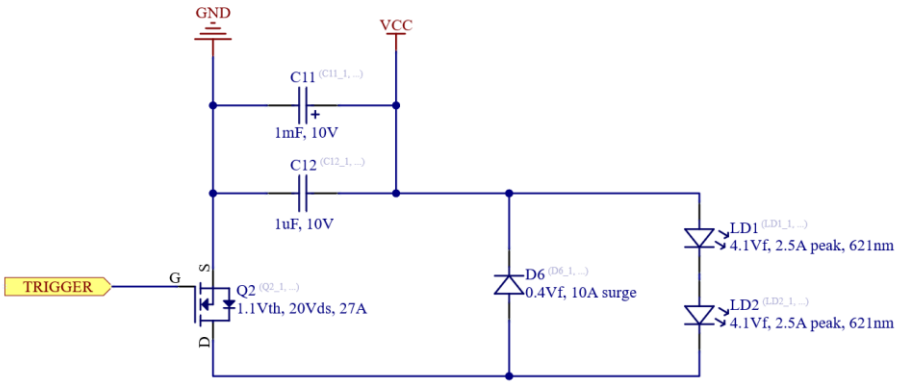


Figure 3.16. LED circuit (repeated 20 times)

The LED schematic (Figure 3.16) was repeated 20 times in the design software, for a total of 40 LEDs on the final PCB. This results in a peak power of $^{40}/_{2} \cdot 6.5\ \mathrm{V} \cdot 7.6\ \mathrm{A} = 988\ \mathrm{W}$, while the average consumption of the board in only around $1\ \mathrm{W}$ (as measured by a bench power supply). Therefore, the designed pulsing light PCB provides high light intensity with optimal energy usage compared to common always-on ring lamps. The PCB design emulates a ring lamp, with LEDs placed regularly around the central camera viewing hole, albeit without a diffuser as it was not found to be necessary for dry conditions. A diffuser is likely to help in wet conditions, however, as glare and the resulting uneven frame lighting caused the system to malfunction. (Renders of the final PCB: Figure 3.17 & Figure 3.18.)

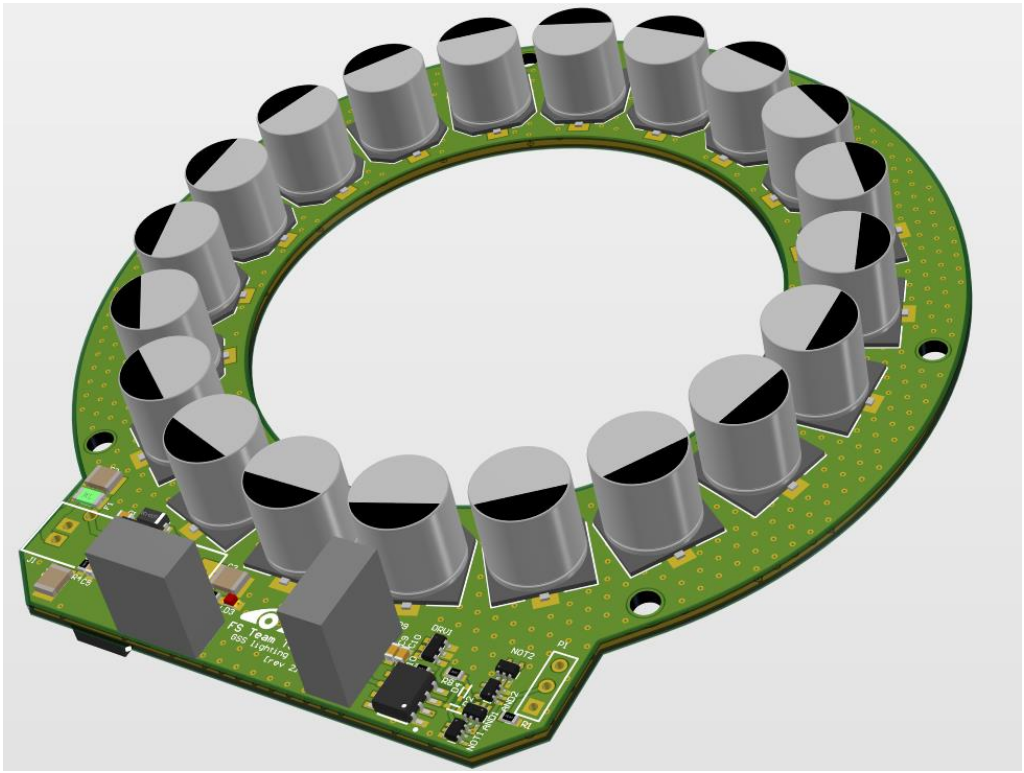


Figure 3.17. Lighting PCB top side

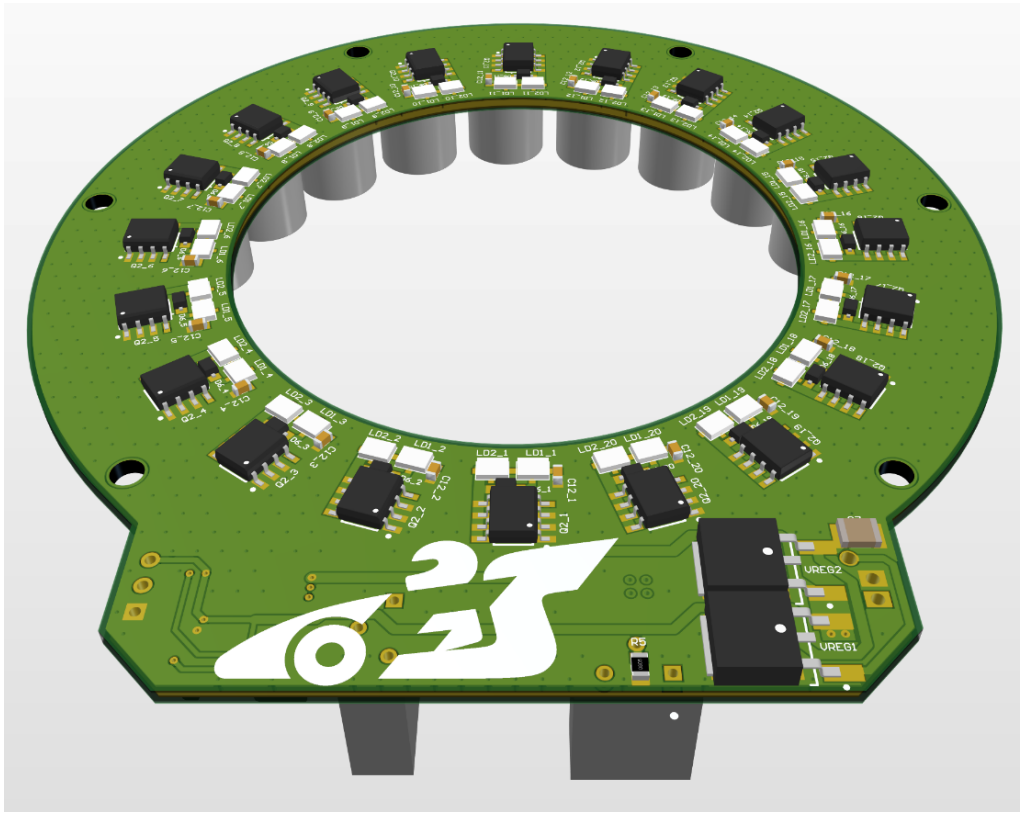


Figure 3.18. Lighting PCB bottom side

3.6 Final system

3.6.1 Housings

The camera was mounted along the middle of the car's longitudinal axis, near the rear axle, on the back wall of the car. The back wall supplied a highly rigid mounting point and a hole integrated into the diffusor for line of sight with the ground. A high-rigidity carbon-fiber clamp was designed to mount the camera steadily to the back wall (Figure 3.19). Additionally, a 3D printed housing for the camera and lighting system was designed, to provide water and dirt resistance for the parts mounted outside the body of the car (Figure 3.20; note the render also includes the non-3D-printed camera mount). To ensure the housing is waterproof, all cables were sealed with a strong epoxy into the 3D printed housings and a sealant was applied to seams between the different housings.

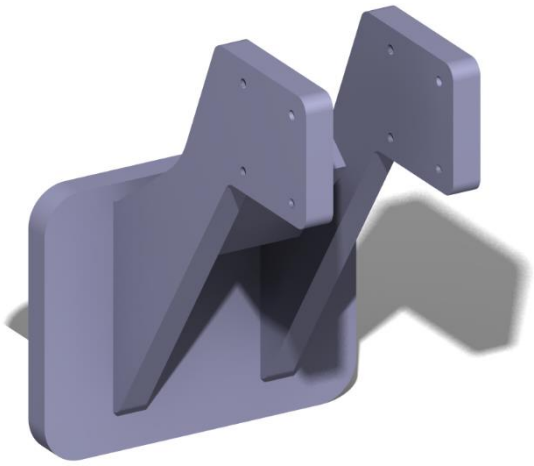


Figure 3.19. Camera mount

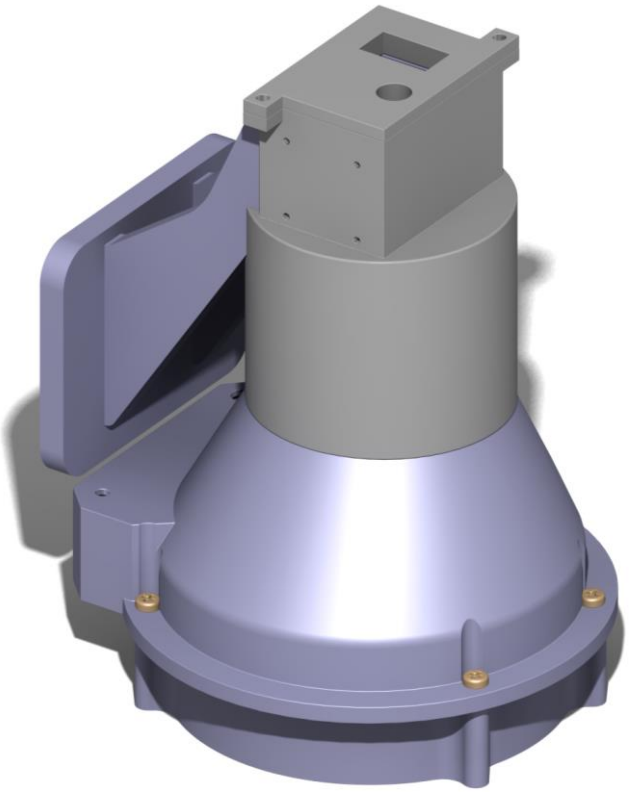


Figure 3.20. Full camera & lighting system housing/mount

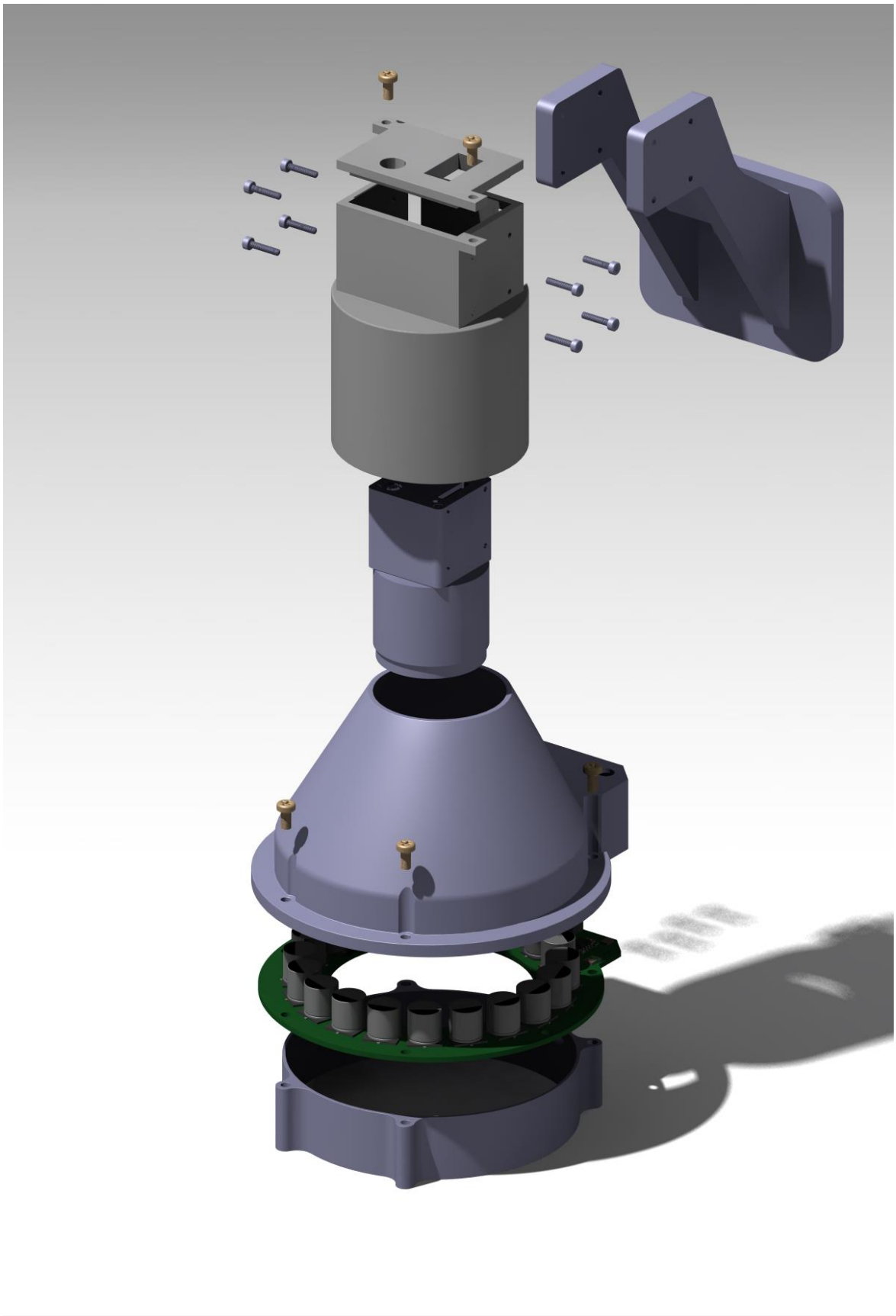


Figure 3.21. Exploded view of the full camera & lighting system housing/mount

A 3D printed housing was also designed for the internally mounted Jetson computer. The housing was designed to be mounted on the inner side of the back wall – near the camera mount. An important aspect of the Jetson housing was to be easily removable to allow for servicing of the car's other systems through the back wall opening. To achieve this a 2-part sliding solution was developed: the top housing containing the Jetson itself could be slid and wedged into place onto the lower part which would be mounted to the car with bolts.

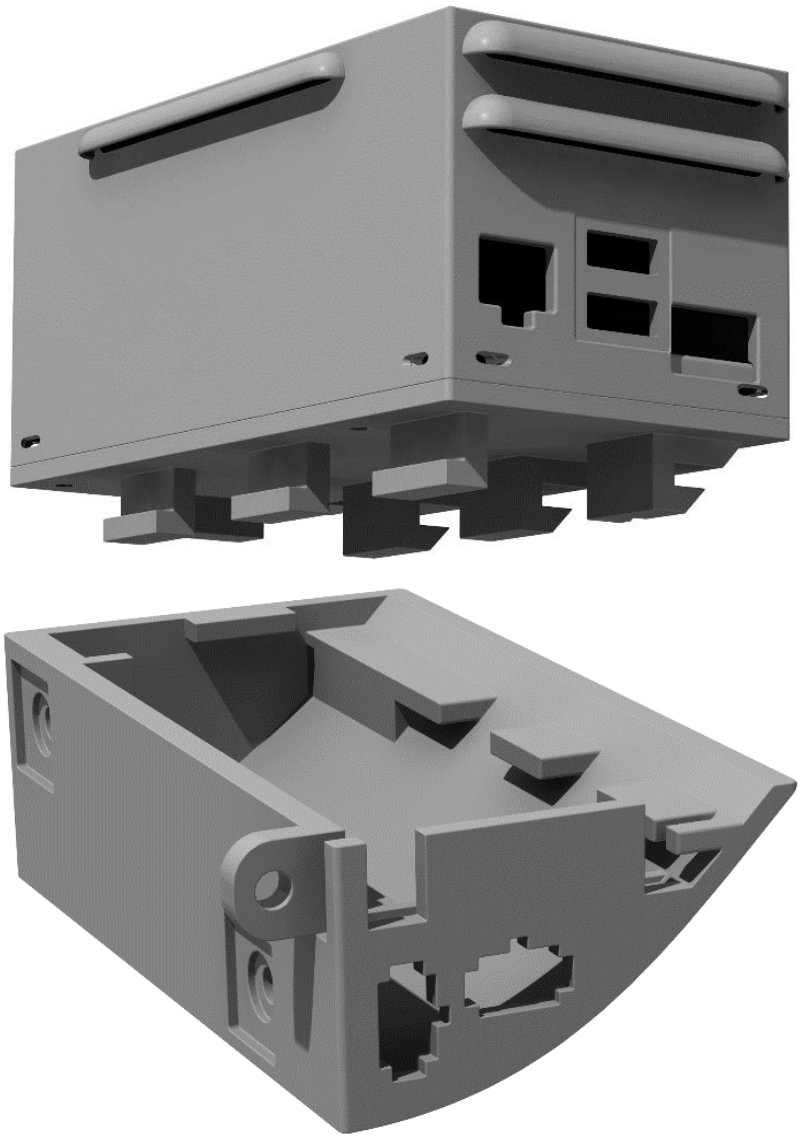


Figure 3.22. Jetson computer housing & mount

3.6.2 Calibration

To eliminate any possible error from deviations in mounting height, the height of the camera and lens must be calibrated in the car's resting position once mounted. Any deviation from the resting position is then compensated by the height sensor. The result of the calibration is a function, f, which maps the height sensor reading, h, (representing the camera's height) to a corresponding pixel size, px_{size}, value: $f(h) = px_{size}$. Calibration should be done on flat ground, to ensure consistent measurements. The method of calibration used is as follows:

1. In the car's resting position, height measurement results are saved for > 5 min, resulting in a median value of h_{med_1}.

2. The physical size of a pixel in the first position, px_{size_1}, is found by measuring the width of the frame at the ground level, w_{frame_1}, with a ruler and applying formula (3.4).

3. Platforms of equal height are placed under each wheel.

4. In the new position, steps 1 and 2 are repeated, resulting in measurement results h_{med_2}, px_{size_2} (and w_{frame_2}).

5. Two points are constructed: (h_{med_1}, px_{size_1}) and (h_{med_2}, px_{size_2}). A linear relationship exists between h_{med} and px_{size}, therefore the function $f(h)$ is given by a line passing through these points with the additional constraint of $h \geq 0$.

The resulting function was directly implemented into the GSS program, as the last line of `getPixelSize` function in Code 3.1. (See chapter 4. Software for more details about the software.)

Code 3.1. Proximity sensor calibration function

```
// Calibration function
double getPixelSize(int range = -1) { // in mm - REQUIRES CALIBRATION
       if (range == -1) range = lastGSSRange; // use saved proximity sensor reading
       if (OPTIONS.RANGE_ENABLED == 0) range = 38.9; // proximity sensor disabled

       return (double)range * 0.00207043 + 0.04952005;
}
```

3.6.3 Accuracy

The accuracy of the proposed GSS relies on template matching working consistently and accurately. Determining whether this is the case is not a simple task and is outside the scope of this thesis. A baseline for the sensor's accuracy can however be given based on the error caused by the height sensor and the resolution of the camera. For the following calculation it is assumed that template matching is always perfectly accurate. Also noteworthy is the fact that the proportional error from the height sensor could be eliminated by e.g. utilizing a telecentric lens, which makes the height sensor redundant.

$$err_{GSS} = \pm\left(\frac{v_{\min}}{2} + \left|err_{\%_{height}}\right|\right) = \pm\left(0.0833\,{}^{\mathrm{km}}/_{\mathrm{h}} + \ 1.33\ \%\right) \tag{3.10}$$

err_{GSS} – GSS system total error,

$v_{\min}$ – resolution of the camera, $\left[{}^{\mathrm{km}}/_{\mathrm{h}}\right]$,

$err_{\%_{height}}$ – the proportional error caused by the height sensor, $[\%]$.

4. Software

The main GSS program was written in C++17 and utilizes OpenCV 2.4.13 from Nvidia's JetPack 2.3.1 all-in-one software development package, which was installed on the Jetson TX1. JetPack 2.3.1 includes Nvidia's L4T (Linux For Tegra) OS image with a file system derived from Ubuntu. [31] The overall experience of Nvidia's L4T was similar to desktop Ubuntu, as all packages desired during development had been precompiled to ARM and were available from the included APT (Advanced Packaging Tool) package sources. The Jetson was remotely accessed over SSH (Secure Shell) and VNC (Virtual Network Computing) for writing, building and testing/debugging the main program. Development was done using the Geany IDE (Integrated Development Environment) and JetPack's included gcc version was used to compile the program on the Jetson itself.

4.1 Command-line interface, configurability

The GSS main program is launched from the command line and to ease testing different configurations on the car, most options (see Code 4.3) were made to be configurable from the program's command line interface. Aside from testing, this also allows the program to be run from a script with the desired options saved within the script. For use during competitions, a `systemd` service was created to run a GSS main program launch script. The launch script disables unnecessary services, which were determined to use the CPU otherwise, to improve performance and launches the main program with the saved command line options.

Code 4.1 GSS launch script

```
#!/bin/sh
bash /home/ubuntu/GSS/src/scripts/stop-everything.sh
while : ; do
nice -n -20 \
/home/ubuntu/GSS/src/scripts/emc_wrapper.sh \
    /home/ubuntu/GSS/src/main \
        -main_stdout_lvl 20 \
        -stdout_lvl 60 \
        -log_lvl 20 \
        -dbg_time \
        -noimg \
        -saveevery 10 \
        -saveimg 0 \
        -get_pics_delay 2300 \
        -can 1 \
        -cropimg 0 \
        -gain 6 \
        -range 0 \
        -range_use 2 \
        "$@"; done
```

Code 4.2 "`stop-everything.sh`"

```
#!/bin/sh

# Included in JetPack
/home/ubuntu/jetson_clocks.sh

service x11vnc stop
service lightdm stop
service rsyslog stop
service systemd-journald stop
```

Code 4.3. GSS program help command line flag (-?) output – all possible command line flags

```
                  ===== COMMAND LINE OPTIONS =====
-noimg                          - no image window
-showevery <i>                  - set image show frequency to every n-th image
-saveimg <i>                    - n=1: save images;        n=2: save image pairs
-saveevery <i>                  - set image save frequency to every n-th image/pair
-range <0...3>                  - amount of range sensors
-range_use <0...2>              - 0 - all sensors, 1 - bus #0, 2 - bus #2
-singe_loops                    - waits for new line (enter key) before starting each template matching thread fn loop

                  ===== CAN OPTIONS =====
-can <0...3>                    - 0: CAN disabled, 1: GSS output, 2: GSS output & acc input, 3: acc input & all output
-can_output 4*<0/1>             - 1@0 debug msg, 1@1: separate messages, 1@2: measured speed, 1@3: calculated speed
-can_rx_id <i>                  - CAN acceleration message ID
-can_tx_id <i>                  - CAN output message ID
-can_tx_id_calc <i>             - CAN output message ID for calculated speeds
-can_tx_id_dbg <i>              - CAN debug message ID (default %i)
-can_acc_multi <d>              - received acceleration multiplier for template estimation

                  ===== DEBUGGING OPTIONS =====
-dbg_all                        - enables all dbg_ flags
-dbg_time                       - log the time taken by various operations
-dbg_sig                        - log multithreading signal changes (also logs signal wait times)
-dbg_sig_wait                   - only log multithreading signal wait times
-dbg_timing                     - log when various multithreaded operations are started
-dbg_can                        - log received/sent can messages
-dbg_range                      - log range sensor info

                  ===== LOGGING OPTIONS =====
                     LOG LEVELS: [10: CRIT, 20: ERR, 30: WARN, 40: LOG, 50: INFO, 60: DBG]
-logdir <str>                   - set the base log directory (MUST be an abolute path!)
-log_lvl <i>                    - max log level written to log files
-stdout_lvl <i>                 - max log level printed to terminal
-main_stdout_lvl <i>            - if set, then log_lvl will be changed to this value once program initialization ends and
the main loop begins
-log_sec_pre <i>                - set log sub-second prescision
-log_hide_line_nr               - source code line numbers will not be added to log messages
-log_hide_func_name             - source file and function names will not be added to log messages

                  ===== CAMERA OPTIONS =====
-cam_timeout <i>                - camera image aquisition timeout in milliseconds
-exp <66...>                    - camera exposure time in microseconds
-gain <0.0-6.9>                 - camera analog gain in dB
-get_pics_instantly             - frame acquisition will NOT start until template matching is completed
-get_pics_delay <i>             - frame acquisition will start after <i> microsec from the start of template matching
-cam_reset                      - reset the camera at the beginning of the program
-cam_noflip                     - disable horizontal image flipping

                  ===== DATA LOGGING OPTIONS =====
-log_acc <0/1>                  - log acceleration received from CAN
-log_spd <0...2>                - 1: log GSS speed, 2: log speed from GSS and speed calculated for acceleration
-log_angle <0/1>                - log calculated car angle

                  ===== IMAGE CROPPING OPTIONS =====
-cropimg <0/1>                  - disable/enable image cropping based on current speed (reducing the template matching area)
-crop_stop <Ld>                 - cropping cutoff speed, based on GSS speed  [km/h]
-crop_start <Ld>                - cropping restart speed, based on wheel speed  [km/h]
-max_acc <d>                    - in g; worst-case acceleration value for cropping in case no CAN msg is received
-crop_err <i>                   - in pixels; extra size added to the cropbox
-crop_t_err <d>                 - in pixels / ms; extra size added to the cropbox for every ms since last GSS measurement

                  ===== TEMPLATE MATCHING OPTIONS =====
-match_method <0-5>             - openCV template matching method
-templ_x <i>                    - template x coordinate
-templ_y <i>                    - template y coordinate
-templ_rect_x <i>               - template rectangle size along the x axis
-templ_rect_y <i>               - template rectangle size along the y axis
```

4.2 Data output – CAN bus

The CAN bus protocol was chosen as the communication interface to be implemented in the prototype GSS. This choice was made, since the main communication bus in the Formula Student Team Tallinn cars is a CAN bus and the CAN bus is commonly used in other automotive applications as well. It must be noted, however, that owing to the general-purpose nature of the Nvidia Jetson platform, many different interfaces are natively supported. It would therefore be trivial and require no additional hardware to support communication over any of the supported interfaces, including: Ethernet, WiFi, Bluetooth 4.0, USB 2.0/3.0, UART, SPI, I^2C [32]. Communication is implemented using the open source SocketCAN driver included in the Linux kernel.

4.3 Device-local logging for debug purposes

During development, it is desirable to log values additional to the outputted measurement results of the sensor to ease debugging in real-life test scenarios where using a debugger is impractical. The C++ standard library does not include any logging utilities, but the open-source peer-reviewed Boost library set does include a library, Boost.Log, for this purpose [33]. However, the Boost.Log library does not itself implement a simple way of logging the file name and line number of the source code which resulted in the logged message. It was decided to implement a simple logging interface with such functionality instead of using workarounds for the Boost.Log library. A single compiler macro is used to implement the logging of the caller's line number, source file name and function:

Code 4.4. Logging macro and function prototype ("log.hpp")

```
#define printLog(...) _printLog(__FUNCTION__, __FILE__, __LINE__, __VA_ARGS__)
void _printLog(const char* FUNC, const char* FILENAME, int LINE,
               int logLevel, const char* frmt, ...);
```

The `printLog` function macro emulates and extends the functionality of the standard `printf` function by accepting a log level indicator, a printf-style format string and any number of arguments beyond it. The `_printLog` function called by the macro integrates with global settings and allows the minimum log level of messages to be print to console/logged to file to be (separately) set via command line arguments (A2.1). Due to the simple configurability, the `printLog` function is used for both debug logging and standard real-time operation monitoring. For an excerpt of the debug log generated for a single template matching cycle see A2.2.1.

4.4 Program loop

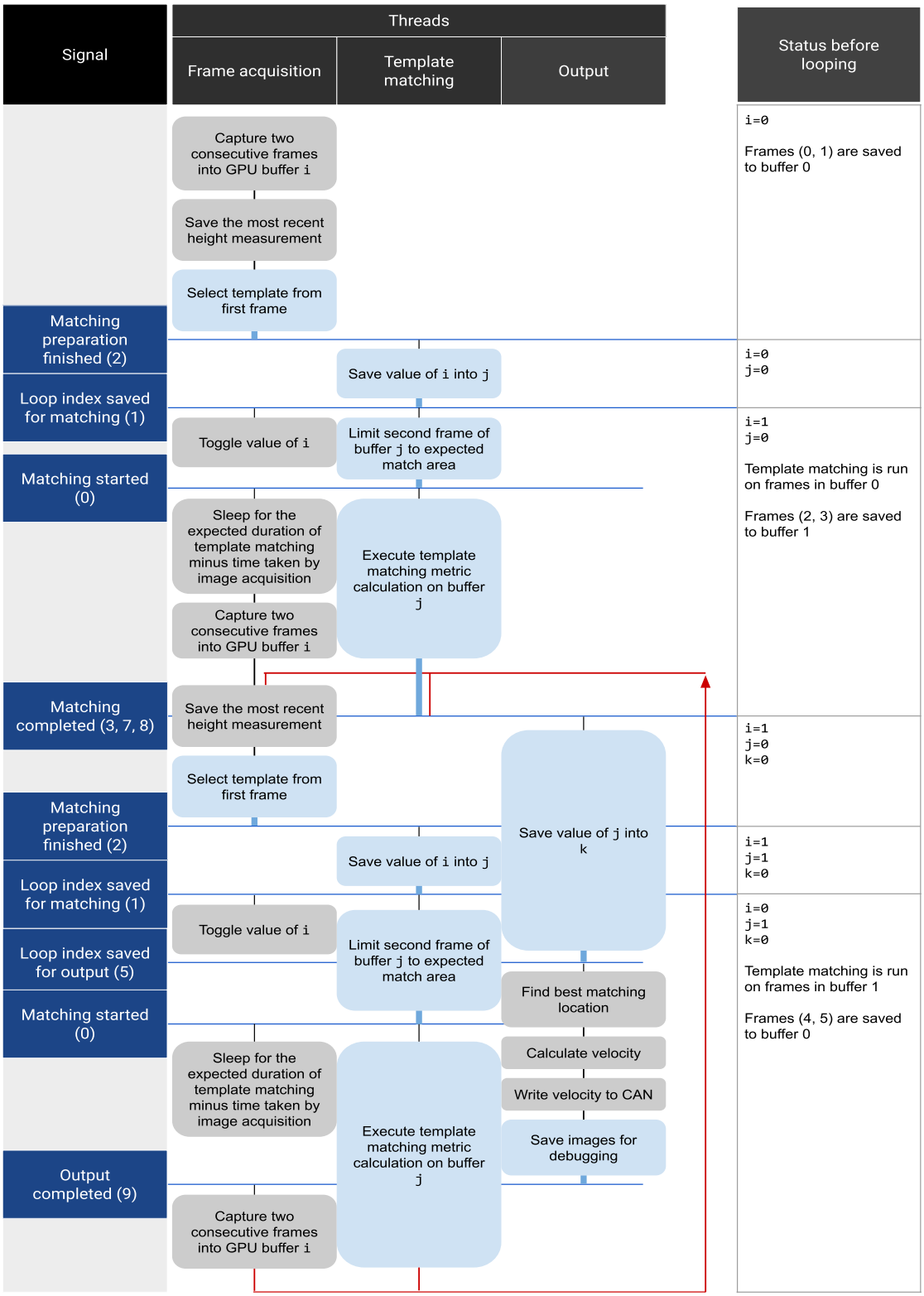
The main GSS program consists of a main thread, which checks/initializes the required devices, starts 6 processing threads and thereafter only handles displaying captured images for real-time debugging if configured to do so from the command line (A2.3). The started threads are: frame acquisition (from camera), template matching, output (calculates the velocity and writes it to CAN/logs), CAN reader and 2 height sensor threads. The CAN reader and height sensor threads operate independently from the other threads, reading data in whenever it becomes available. The other 3 threads are tightly interwoven to minimize unnecessary waiting and therefore improve performance.

C++ mutex locks were used to synchronize the threads in key checkpoints to ensure the correct order of execution for the algorithm is always followed. These were implemented as "signals": a thread would set a specific signal when a key action was preformed meanwhile other threads could wait for the signal to be set before continuing execution. Using standard mutex locks for this avoids busy-waiting. In Table 4.1, a high-level representation of the algorithm implemented on the 3 threads is given, in top-to-bottom order. Rounded rectangles represent an action of a given thread; actions connected with lines are executed consecutively. Light-blue actions indicate that a signal is set after the action completes. Actions beginning from a signal line wait for that signal to be set before starting. The red arrow represents the part of the algorithm which is looped indefinitely.

The main goal of the described algorithm is to keep the GPU constantly busy running the template matching algorithm. Frames for the next template matching execution are acquired just-in-time for the completion of template matching on the previous frames: if they were captured after template matching finishes, a $^{1}/_{\text{fps}} = 2\ \text{ms}$ delay would be introduced; if they were captured too early, the output velocity signal latency would increase. Output is also handled by a separate thread, as it handles multiple slow operations: calculating the velocity requires the minimum/maximum to be found from the template matching output array and saving the frames for later debugging means a transfer from RAM to disk is required.

Implementing the velocity measurement itself is trivial: the original location of the template is pre-determined and therefore known; the time shift between 2 frames is fixed by the camera settings ($2\ \text{ms}$ in the case of a $500\ \text{fps}$ camera). After template matching is completed and the template's new location is found, the shift in location within $2\ \text{ms}$ can be found by subtracting the original template location. The location shift is multiplied by the value given by the calibration function described in 3.6.2 and divided by the time shift to find the velocity. A 3-value median filter is then applied to the velocity readings before being output to the car's CAN network.

Table 4.1. Main algorithm of the GSS program



4.5 Camera integration

For interaction with the camera, Ximea's xiAPI C++ API was used. The camera is first initialized with default or command-line defined parameters and a software trigger signal is issued to the camera to capture 2 consecutive frames. The frames are first stored in a CPU RAM buffer and copied over to GPU RAM. As the Jetson TX1 does not have dedicated GPU memory, the same physical RAM is shared by both the CPU and GPU and a zero-copy approach is theoretically possible by setting a CUDA flag: `cudaSetDeviceFlags(cudaDeviceMapHost)` [34, 35]. This approach was not successfully implemented and the CPU-to-GPU memory copy was retained. A memory optimization that was made, however, consists of pre-allocating and reusing the same GPU memory buffers. The buffers are initialized as OpenCV's `cv::gpu::GpuMat` objects and `cudaMemcpy` is used to copy data directly from the xiAPI camera buffers to GPU memory.

Code 4.5. Example of direct copy from xiAPI XI_IMG to OpenCV GpuMat

```
int image_size; XI_IMG xi_img; int frame_height = 488; int frame_width = 648;


xiGetParamInt(xiH, XI_PRM_IMAGE_PAYLOAD_SIZE, &image_size);
cv::gpu::GpuMat gpu_mat = cv::gpu::GpuMat(frame_height, frame_width, CV_8U, gpu_mat);
cudaMemcpy(gpu_mat.data, xi_img.bp, image_size, cudaMemcpyHostToDevice);
```

4.6 Height sensor integration

For the VL6180X sensor, the API needed to be ported to the Jetson platform. The API provided by STMicroelectronics implements most high-level functions of the sensor in C, leaving the low-level device specific I^2C access to be implemented by the developer. On the Jetson TX1, the I^2C interfaces can be accessed by opening the desired device file under `/dev/`: `/dev/i2c-0` through `/dev/i2c-5` exist, but on the Auvidea J120 carrier only `/dev/i2c-0` and `/dev/i2c-1` are externally accessible. The 2 buses were used to connect up to 2 sensors simultaneously using the same device address, which required additional modification of the VL6180X API. Additionally, the API was ported to C++ to simplify interoperability with the VL6180X code. For calibration (as described in item 3.6.2), a simple program to log and average the values over a long time was created.

4.7 Template matching optimization: base image masking

A common approach to improving template-based matching performance is downscaling the images and running template matching first on the low-resolution images to get an approximate match. For a precision match, template matching is run again on the original resolution images, but with the base image cropped to the area of the previous approximate match. Given the nature of regular velocity measurements within the proposed application for template matching, the first template matching run can be eliminated. Cropping the base image to an area around the previous match gives a similar result for minimal computational effort.

It should be possible to integrate the acceleration reading of an accelerometer mounted on the car to be measured to very accurately determine where the template should have moved within the time elapsed from the last template matching execution, as the dead reckoning error should not accumulate considerably during the ~ 10 ms timeframe. The accelerometer on the FEST19 car was too noisy, however, due to a poor non-rigid mount and this method was not used. Instead, a simpler method of considering all possibilities and expanding the match area in every direction was used. To determine the size of the final cropped image, a small constant expansion and an expansion based on an assumed acceleration of 3 g starting from the previous frame capture time was added to the match area. While this utilizing the car's accelerometer or assuming a smaller acceleration for accelerating than braking could further improve performance, the template matching base image size was still reduced considerably compared to the original resolution of the frame, as seen in Figure 4.1 (the "Expected match location" is what the base image was cropped to).

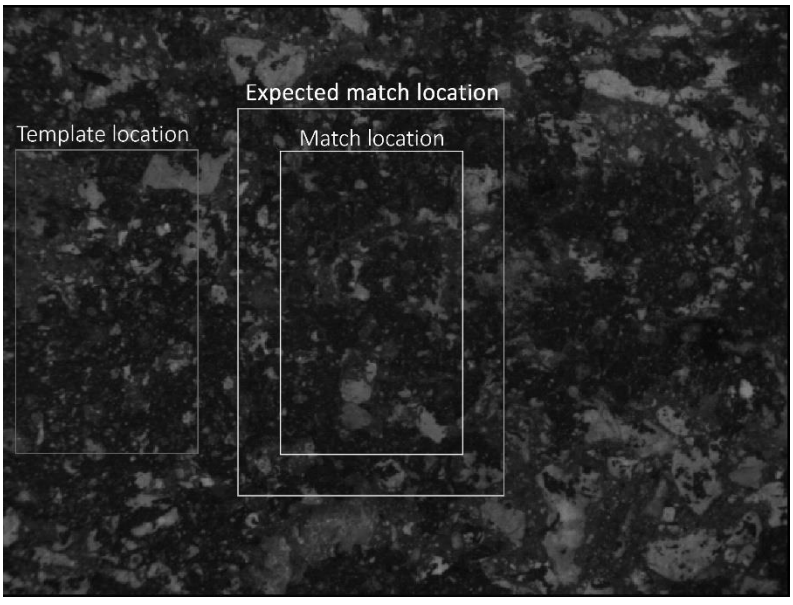


Figure 4.1. Template matching frame

5. Results

The proposed GSS was integrated into the control system of the FEST19 car to provide slip control. With the velocity reading provided by the GSS, the ideal no-slip rotational velocity for each wheel could be calculated and wheel slip could always be limited to $\sim 10\%$ above it. The original slip controller had no reliable feedback available and therefore needed manual tuning, which would only be accurate so long as external conditions, such as tire temperature, track surface & humidity, do not change. The slip controller with feedback from the template matching GSS was successfully used in the 18/19 season's competitions. Due to time constraints and issues with the height sensors, further torque control integration was not attempted during the 18/19 season.

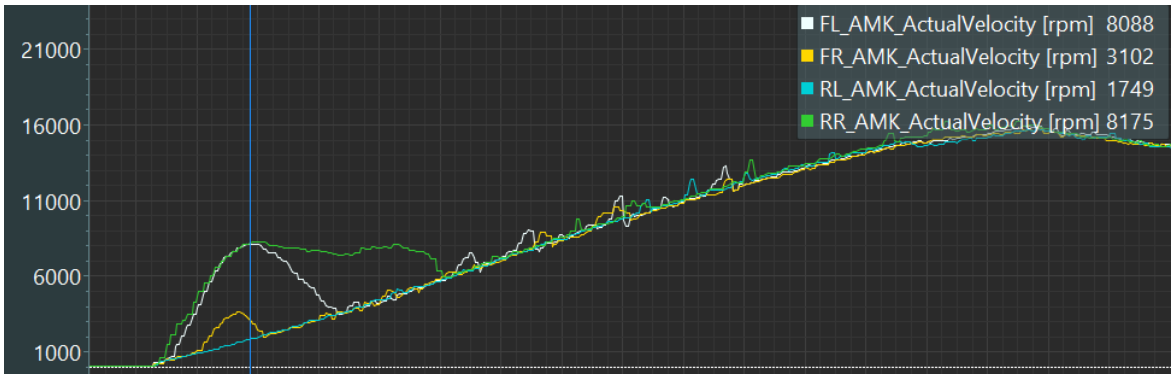


Figure 5.1. Acceleration with a poorly tuned open-loop slip controller

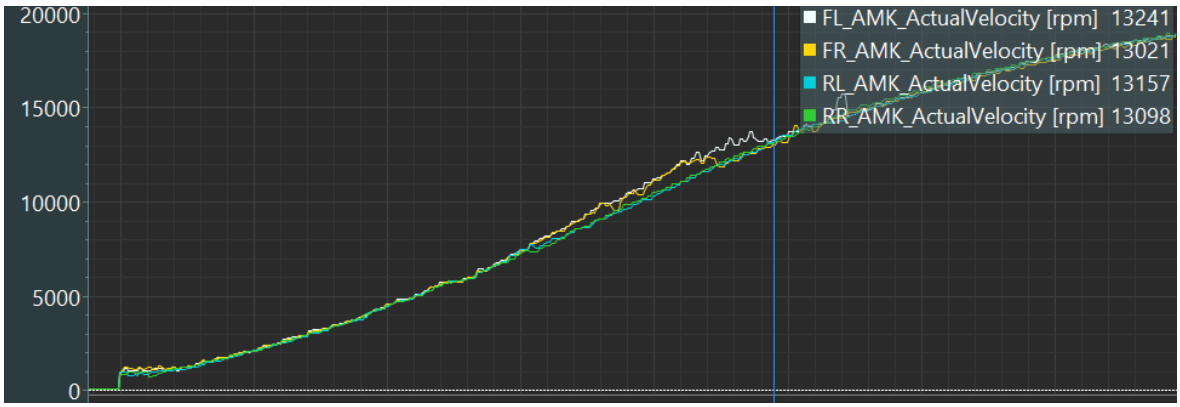


Figure 5.2. Acceleration with a slip controller utilizing the proposed GSS

Summary

Machine vision provides a potential alternative to the approaches of existing commercial high-performance non-contact ground speed sensors. A template matching GSS can be built from mostly off-the-shelf components, with the specific choice of component parameters directly influencing the final performance and cost of the GSS. The theory of operation of the proposed GSS was described along with the selection criteria for the sensor's components. A working template matching GSS was developed, integrated into the Formula Student car and utilized for slip control feedback in official competitions.

Along with the off-the-shelf components, a custom lighting solution was developed to limit the amount of natural sunlight reaching the camera and generate short intense pulses of light to overcome the camera's limitations and achieve shorter exposures than otherwise possible. Additionally, an easily configurable software solution was developed and optimized. An efficient multi-threaded algorithm was designed to maximize utilization of the embedded computer's GPU. An alternative approach to the two-step low-resolution→high-resolution template matching optimization technique was used, considerably limiting the area to be processed by template matching with only a single template matching run.

Some aspects were not fully explored within the context of this thesis and further research needs to be done to determine:

- The benefits/disadvantages of using a telecentric lens.
- An accurate height sensing mechanism, e.g. by utilizing a car's suspension travel sensors.
- The accuracy and repeatability of the template matching algorithm itself, to determine the final accuracy of the proposed GSS.
- Feasibility of zero-copy memory operations.
- Feasibility of cropping the template matching base image based on integrated input from accelerometers.
- The proposed sensor's suitability for applications other than slip control.
- Feasibility of a higher-performing template matching GSS by utilizing newer/more expensive components.

Kokkuvõte

Masinnägemine pakub potentsiaalset alternatiivi turul eksisteerivatele suure jõudlusega kontaktivabadele maapinna kiiruse anduritele. Malli sobitamisel töötava anduri saab luua suures osas valmiskomponentidest, kusjuures anduri lõpliku jõudluse ja hinna saab määrata komponentide valikuga. Töös kirjeldati malli sobitamisel töötava anduri tööpõhimõtet ja komponentide valiku kriteeriumeid. Arendati välja töötav andur, mis integreeriti Tudengivormeli FEST19 vormelisse ja mida kasutati edukalt ametlikel võistlustel rehvide libisemise piiramiseks.

Valmiskomponentidele täiendavalt loodi valgustussüsteem kaamerasse jõudva loomuliku päikesevalguse piiramiseks ja lühikeste intensiivsete valgusimpulsside genereerimiseks, eesmärgiga parandada kaamera vähimat võimalikku säriaega. Samuti loodi lihtsasti konfigureeritav ja optimeeritud tarkvaralahendus. Töötati välja efektiivne paralleelselt jooksutatav algoritm sardsüsteemi graafikavõimendi kasutuse maksimeerimiseks. Kahesammulise madala resolutsiooniga→kõrge resolutsiooniga malli sobitamise optimeerimismeetodi asemel kasutati ühesammulist ja malli sobitamisega töödeldavat ala oluliselt vähendavat meetodit.

Mitmed aspektid jäid antud lõputöö kontekstis süvitsi uurimata; täiendavat uurimistegevust oleks vaja läbi viia, et leida/määrata:

- teletsentrilise läätse kasutamise eelised/puudused,
- täpne kõrguse mõõtmise meetod, nt. kasutades auto vedrustusandureid,
- malli sobitamise algoritmi täpsus ja korratavus, et määrata maapinna kiiruse anduri lõplik täpsus,
- kopeerimiseta mäluoperatsioonide teostatavus,
- kiirendusandurite integreeritud lugemi põhjal kaadri kärpimise teostatavus,
- malli sobitamisel töötava anduri sobivus teistes rakendustes (lisaks libisemise piiramisele),
- suurema jõudlusega anduri teostatavus kasutades uuemaid/kallimaid komponente.

References

[1] "f-number," [Online]. Available: https://en.wikipedia.org/wiki/F-number. [Accessed 2020-05-23].

[2] "Entrance pupil," [Online]. Available: https://en.wikipedia.org/wiki/Entrance_pupil. [Accessed 2020-05-03].

[3] Wikipedia, "Quantum efficiency," [Online]. Available: https://en.wikipedia.org/wiki/Quantum_efficiency. [Accessed 2020-05-18].

[4] VBOX Automotive, "A Comparison of non-contact speed sensors," 2005-05. [Online]. Available: http://www.racelogic.co.uk/_downloads/vbox/Application_Notes/Comparison%20of%20Non-contact%20Speed%20Sensors.pdf. [Accessed 2020-04-20].

[5] VBOX Automotive, "Speed Sensor Range (VBSS05 – VBSS100_V4G)," 2018-06-13. [Online]. Available: https://www.racelogic.co.uk/_downloads/vbox/Datasheets/Speed_Sensors/RLVBSS_DATA.pdf. [Accessed 2020-04-20].

[6] Kistler, "Correvit SFII sensors," 2019-02. [Online]. Available: https://www.kistler.com/?type=669&fid=86251&model=document. [Accessed 2020-04-20].

[7] DICKEY-john, "Radar II," [Online]. Available: http://www.dickey-john.com/product/radar-ii/. [Accessed 2020-04-20].

[8] VBOX Automotive, "VBOX Speed Sensors | 5 - 100Hz," [Online]. Available: https://www.vboxautomotive.co.uk/index.php/en/products/sensors/speed-sensors#specs. [Accessed 2020-04-20].

[9] Kistler, "Instruction Manual Correvit SFII Non-Contact Optical Sensors," 2016-12. [Online]. Available: https://www.kistler.com/?type=669&fid=86249&model=document. [Accessed 2020-04-20].

[10] Sensor-1, "DJ-RVSII | DICKEY-john Radar II," [Online]. Available: https://sensor-1.com/products/dj-rvsii-dickey-john-radar-ii. [Accessed 2020-04-20].

[11] FXTOP, "Historical Converter," [Online]. Available: https://fxtop.com/en/historical-currency-converter.php?A=798&C1=USD&C2=EUR&DD=20&MM=04&YYYY=2020&B=1&P=&I=1&btnOK=Go%21. [Accessed 2020-04-20].

[12] Zauba Technologies Pvt Ltd, "Detailed Import Data of VBOX," [Online]. Available: https://www.zauba.com/import-vbox/hs-code-90318000/ip-INMAA4/p-1-hs-code.html. [Accessed 2020-04-20].

[13] FXTOP, "Historical currency converter," [Online]. Available: https://fxtop.com/en/historical-currency-converter.php?A=554548&C1=INR&C2=EUR&DD=14&MM=01&YYYY=2016&B=1&P=&I=1&btnOK=Go%21. [Accessed 2020-04-20].

[14] Formula1_, "Reddit," 2019-07. [Online]. Available: https://www.reddit.com/r/formula1/comments/c6mz3c/bottas_fp2_crash_info_on_the_parts_dangling_out/. [Accessed 2020-04-20].

[15] D. Chetverikov, "Template matching and feature detection (part 1)," 2001-07. [Online]. Available: http://www.inf.u-szeged.hu/ssip/2001/handouts/chetverikov/chetverikov1.pdf. [Accessed 2020-04-28].

[16] R. N. Luces, "Template-based versus Feature-based Template Matching," RAX Automation Suite, 2019-11-16. [Online]. Available: https://medium.com/datadriveninvestor/template-based-versus-feature-based-template-matching-e6e77b2a3b3a. [Accessed 2020-04-28].

[17] Open Source Vision Foundation, "Feature Matching + Homography to find Objects," 2020-04-03. [Online]. Available: https://docs.opencv.org/4.3.0/d1/de0/tutorial_py_feature_homography.html. [Accessed 2020-04-28].

[18] Open Source Vision Foundation, "Template Matching," 2020-04-03. [Online]. Available: https://docs.opencv.org/4.3.0/de/da9/tutorial_template_matching.html. [Accessed 2020-04-28].

[19] scikit-image.org, "Template Matching," [Online]. Available: https://scikit-image.org/docs/0.16.x/auto_examples/features_detection/plot_template.html. [Accessed 2020-05-04].

[20] M. Larabel, "Raspberry Pi 3 Benchmarks vs. Eight Other ARM Linux Boards," Phoronix Media, 2016-03-05. [Online]. Available: https://www.phoronix.com/scan.php?page=article&item=raspberry-pi-3&num=3. [Accessed 2020-05-11].

[21] Open Source Vision Foundation, "CUDA," [Online]. Available: https://opencv.org/platforms/cuda/. [Accessed 2020-05-11].

[22] NVIDIA Corporation, "Buy the Latest Jetson Products," [Online]. Available: https://developer.nvidia.com/buy-jetson. [Accessed 2020-05-11].

[23] Auvidea GmbH, "J120 & J121 technical reference manual," 2019-07. [Online]. Available: https://auvidea.eu/download/manual/J120/J120_J121_technical_reference_2.0.pdf. [Accessed 2020-05-11].

[24] M. Isted, "Pixhawk 2 with Jetson TX2 Build," 2018-08-23. [Online]. Available: https://mikeisted.wordpress.com/2018/08/23/pixhawk-2-with-jetson-tx2-build/. [Accessed 2020-05-11].

[25] XIMEA, "MQ003MG-CM," 2015. [Online]. Available: https://www.ximea.com/en/products/usb3-vision-cameras-xiq-line/mq003mg-cm. [Accessed 2020-05-15].

[26] CMOSIS bvba, "CMV300 Datasheet - VGA resolution CMOS image sensor," 2016. [Online]. Available: https://ams.com/documents/20143/36005/CMV300_DS000428_1-00.pdf. [Accessed 2020-05-18].

[27] D. Carr, "How To Calculate Field of View In Photography," Shutter Muse, 2016-03-29. [Online]. Available: https://shuttermuse.com/calculate-field-of-view-camera-lens/. [Accessed 2020-05-15].

[28] Computar, "M1214-MP2," 2008-05. [Online]. Available: https://computar.com/resources/files_v2/162/M1214-MP2.pdf. [Accessed 2020-05-16].

[29] STMicroelectronics, "VL6180X Proximity and ambient light sensing (ALS) module," 2016-03. [Online]. Available: https://www.st.com/resource/en/datasheet/vl6180x.pdf. [Accessed 2020-05-16].

[30] M. S. Malik, "Modeling and Simulation of a Novel Module for Thermoelectric Power Generation from Solar Photovoltaic Panels," 2019-06. [Online]. Available: https://www.researchgate.net/publication/333925279_Modeling_and_Simulation_of_a_Novel_Module_for_Thermoelectric_Power_Generation_from_Solar_Photovoltaic_Panels. [Accessed 2020-05-18].

[31] NVIDIA Corporation, "Jetpack 2.3.1 Release Notes," [Online]. Available: https://developer.nvidia.com/embedded/jetpack-2_3_1. [Accessed 2020-05-18].

[32] Nvidia Corporation, "NVIDIA Jetson TX1 System-on-Module Data Sheet," 2016-11-22. [Online]. Available: http://developer.nvidia.com/embedded/dlc/jetson-tx1-module-data-sheet. [Accessed 2020-04-26].

[33] A. Semashev, "Chapter 1. Boost.Log v2," Boost Foundation, 2016. [Online]. Available: https://www.boost.org/doc/libs/1_72_0/libs/log/doc/html/index.html. [Accessed 2020-04-26].

[34] NVIDIA Corporation, "CUDA Zero Copy On TX1," 2017-03. [Online]. Available: https://forums.developer.nvidia.com/t/cuda-zero-copy-on-tx1/48455/2. [Accessed 2020-05-18].

[35] NVIDIA Coropration, "Data types used by CUDA Runtime," [Online]. Available: http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/online/group__CUDART__TYPES_g3762be9cccdd809a4ca128354fd134b0.html. [Accessed 2020-05-18].

[36] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[37] Edmund Optics Ltd, "Understanding Focal Length and Field of View," 2015-03-11. [Online]. Available: https://www.edmundoptics.eu/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/. [Accessed 2020-05-03].

[38] Computar, "M1214-MP2," [Online]. Available: https://computar.com/product/553/M1214-MP2. [Accessed 2020-05-16].

[39] Adafruit, "Adafruit (PID 3316) VL6180X Time of Flight Distance Ranging Sensor (VL6180)," [Online]. Available: https://www.amazon.com/Adafruit-VL6180X-Flight-Distance-Ranging/dp/B01N0ODI3Q. [Accessed 2020-05-16].

[40] Wikipedia, "Full width at half maximum," [Online]. Available: https://en.wikipedia.org/wiki/Full_width_at_half_maximum. [Accessed 2020-05-18].

Appendices

1. Lighting system PCB power schematics

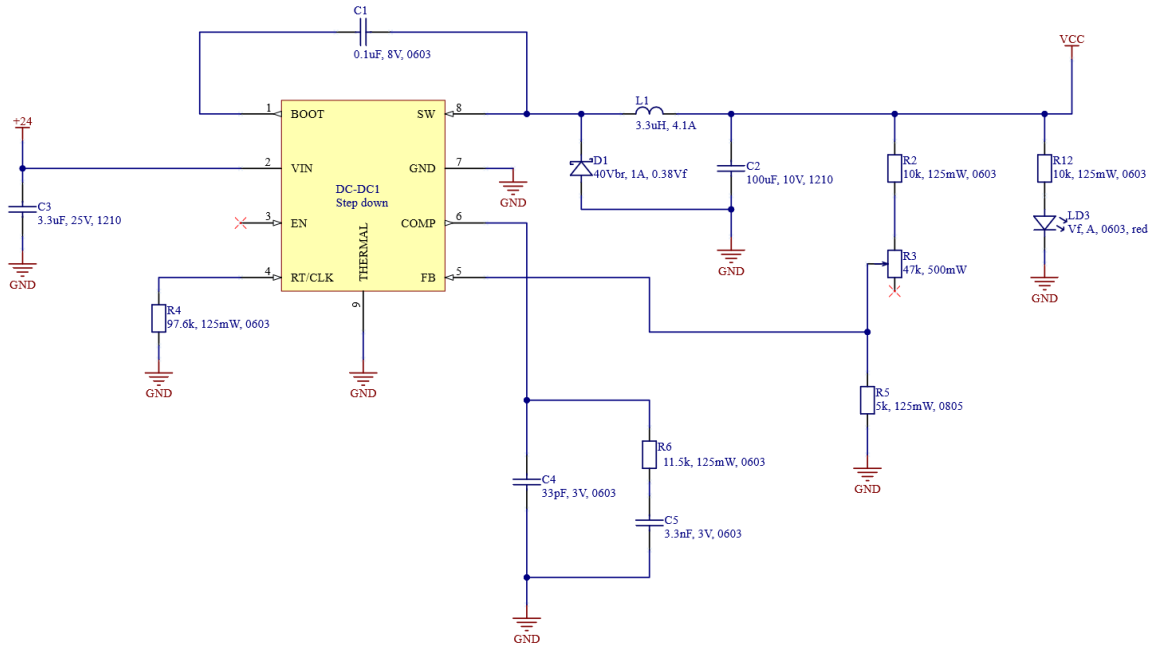


Figure 1.1. Adjustable 5A step down converter – main LED power supply

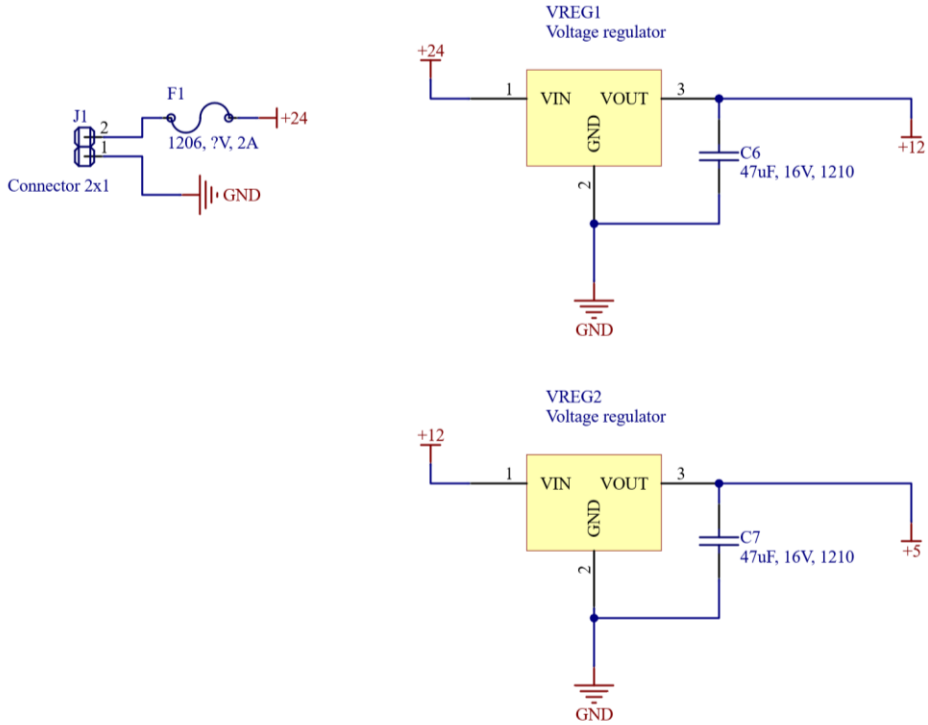


Figure 1.2. Auxiliary logic 5V and 12V regulators

2. Code snippets

2.1 Command line options

Code 2.1. `GlobalSettings` struct from "`options.hpp`"

```
struct GlobalSettings {
        long LOOP_LIMIT = 0;
        bool SINGLE_LOOPS = false;

        /// CAN options
        int CAN_ENABLED = 2;
        int CAN_MODE = (0 << 3) + (0 << 2) + (1 << 1) + 0;
        unsigned int CAN_TX_ID = 0x350; // gss measured speed (mm/s),
        unsigned int CAN_TX_ID_CALC = 1420; // calculated speed
        unsigned int CAN_TX_ID_DBG = 1421; // debug info
        unsigned int CAN_RX_ID = 0x41A; // acceleration
        unsigned int CAN_RX_FL = 0x2E2;
        unsigned int CAN_RX_FR = 0x2D2;
        unsigned int CAN_RX_RL = 0x2A2;
        unsigned int CAN_RX_RR = 0x292;
        unsigned int CAN_RX_ECU1 = 1328;
        double CAN_ACC_MULTI = 2;

        /// Range sensor / I2C options
        int RANGE_ENABLED = 1; // how many sensors there are
        int RANGE_USE = 0; // 0 - all sensors, 1 - bus #0, 2 - bus #2

        /// image output window options
        bool IMAGE_OUTPUT = true;
        int IMAGE_SHOW_RATE = 10;

        /// image saving options
        int IMAGE_LOG = 0; // 0 - off; 1 - save first image, 2 - save image pairs
        int IMAGE_LOG_RATE = 100; // save every <IMAGE_LOG_RATE>-th image pair

        /// image matching options
        double TEMPLATE_ERROR_PER_MS = 0.25;
        double TEMPLATE_ERROR_OFFSET = 5;

        /// picture acquisition options
        bool GET_PICS_SLEEP = true; // whether to sleep for GET_PICS_SLEEP_T mcs after matching
starts to get pics (true) or instead get pics when matching ends (false)
        int GET_PICS_SLEEP_T = 5500; // microseconds

        /// debug options to enable additional data logging
        bool DBG_SIG = false;
        bool DBG_SIG_WAIT = false;
        bool DBG_TIMINGS = false;
        bool DBG_TIME = false;
        bool DBG_CAN = false;
        bool DBG_RANGE = false;
        //bool DBG_TRACKERS = false; //  deprecated

        int LOG_LEVEL = 0;
        int _LOG_LEVEL = DBG; /// log level is set to this after the log files have been created
        int STDOUT_LEVEL = DBG;
        int *MAIN_LOOP_STDOUT_LEVEL = &STDOUT_LEVEL; /// stdout level is set to the value this
points to after program initialization ends (and before the main() while loop begins)
        int _MAIN_LOOP_STDOUT_LEVEL;
        bool LOG_FUNC_NAME = true;
        bool LOG_LINE_NR = false;
        int LOG_SEC_PRECISION = 3;
```

```
        /// data logging options
        bool LOG_ACC = false;
        int LOG_SPD = 1;
        bool LOG_ANGLE = false;

        /// image cropping options
        bool CROP_IMAGE = true;
        long double CROP_STOP_SPD = 75.1;        // [km/h]
        int CROP_STOP_WHL_SPD = 15000; // [rpm]
        //long double CROP_START = 60.1; // [km/h]
        long double MAX_ACC = 3.01;              // [g]
        int CROP_ERR = 5;                        // [px]
        double CROP_T_ERR = 1;   // [px / ms]


        /// template matching options
        int MATCH_METHOD = 1;    // Different methods 0-5
        int TEMPL_X = 10;        // Template location
        //int TEMPL_RECT_X = 190;        // Size of template
        int TEMPL_RECT_X = 150;  // Size of template
        int TEMPL_RECT_Y = 250;
        int TEMPL_Y = (488 - TEMPL_RECT_Y) / 2;


        /// log file paths
        std::string LOG_BASE_DIR = "/home/ubuntu/GSS/logs/"; // Base folder for logs (must be
absolute! ~ does not work)
        std::string SPEED_LOG_PREFIX = "SpeedLog-"; // possibly deprecated

        /// camera options
        /// TODO: more camera options?
        int CAM_TIMEOUT = 100; // milliseconds camera timeout
        int EXPOSURE = 66; //1900 //150 // Camera exposure time in us (Choosing too big exp. time
(> ~2000) will decrease frame rate); 66 is minimum
        float GAIN = 0.; // 0.0 - 6.9 dB
        bool CAM_RESET = false;
        bool CAM_FLIP = true;



        int _initFromArguments(int &argc, char *argv[]);
};

extern GlobalSettings OPTIONS;
```

2.2 Logging solution

Code 2.2. “log.hpp” snippet

```
#ifndef _LOG_H
#define _LOG_H

/// LOG LEVELS
#define CRIT  10
#define ERR   20
#define WARN  30 /// <= WARN goes to ERR_FILE
#define LOG   40 /// > WARN && < INFO goes to LOG_FILE
#define INFO  50 /// >= INFO goes to DBG_FILE
#define DBG   60

/// equivalent to printLog(int logLevel, const char* frmt, ...)
///           <const char* frmt, ...>   -   printf formatting
/// usage example: printLog(CRIT, "Error nr %i occurred: %s", 13, "bad things");
///           prints formatted critical error to stdout and log file (depending on the
settings in OPTIONS)
#define printLog(...) _printLog(__FUNCTION__, __FILE__, __LINE__, __VA_ARGS__)


// DO NOT USE _printLog()! Use printLog() (defined as a macro above) instead!
extern "C" {
       void _printLog(const char* FUNC, const char* FILENAME, int LINE, int logLevel,
const char* frmt, ...);
}

#endif
```

Code 2.3. “log.cpp” snippet

```
#include "log.hpp"

#include <cstring>
#include <sys/stat.h>
#include <cerrno>
#include <cstdarg>
#include <string>

FILE *ERR_FILE, *LOG_FILE, *DBG_FILE;


void _printLog(const char* FUNC, const char* FILENAME, int LINE, int logLevel, const char* frmt,
...) {
        /// if nothing will be logged to anywhere, then quit immediately
        if (OPTIONS.LOG_LEVEL < logLevel && OPTIONS.STDOUT_LEVEL < logLevel) return;

        /// prepare variable arguments
        va_list args;
        va_start(args, frmt);

        /// remove extension from filename
        std::string filename = (std::string)FILENAME;
        filename = filename.substr(0, filename.find("."));

        // resolve log level string if a defined level is used, otherwise display number
        std::string s;
        switch (logLevel) {
                case CRIT:
                        s = "[CRIT]";
                        break;
                case ERR:
                        s = "[ERR] ";
                        break;
                case WARN:
                        s = "[WARN]";
                        break;
                case LOG:
                        s = "[LOG] ";
                        break;
                case INFO:
                        s = "[INFO]";
                        break;
                case DBG:
                        s = "[DBG] ";
                        break;
                default:
                        s = "[" + std::to_string(logLevel) + "]";
        }

        // surround the frmt argument with debug values and then printf (using the encapsulated
frmt as the format string)
        std::string format = getCurrentTime() + " " + s + " " +
                (OPTIONS.LOG_FUNC_NAME ? "{" + filename + "::" + (std::string)FUNC + "}    " : "")+
                (std::string)frmt +
                (OPTIONS.LOG_LINE_NR ? " (@" + std::to_string(LINE) + ")\n" : "\n");
        char logString[1024];
        std::vsprintf(logString, format.c_str(), args);
        if (OPTIONS.STDOUT_LEVEL >= logLevel) printf("%s", (const char*)logString);
        if (OPTIONS.LOG_LEVEL >= logLevel) {
                auto file= (logLevel > WARN ? (logLevel >= INFO ? DBG_FILE : LOG_FILE) : ERR_FILE);
                fprintf(file, "%s", logString);
                fflush(file);
        }
}
```

2.2.1 Example log

Code 2.4. Debug logging output

```
07:05:58.434 [INFO] {can::execWriteToCan}    CAN TX:  0x350  [8]  0x27 0x83 0x00 0x47 0x33 0x2B
0x4D 0x2D
07:05:58.437 [DBG]  {camera::GetPicsBlockingCapture}    Starting to get pics (i: 1)...
07:05:58.438 [INFO] {camera::ExecGetPics}    Camera trigger set in 1.247 ms (0 retries).
07:05:58.438 [INFO] {RangeSensor::readRange}    Range 1 = 40; median/avg: 43 / 45
07:05:58.438 [DBG]  {RangeSensor::readRange}    Range poll measurement completed.
07:05:58.438 [INFO] {threadFn::rangeThreadFn}    Range read in 17.265 ms
07:05:58.438 [DBG]  {RangeSensor::readRange}    Starting range poll measurement...
07:05:58.439 [INFO] {RangeSensor::readRange}    Range 0 = 52; median/avg: 51 / 45
07:05:58.439 [DBG]  {RangeSensor::readRange}    Range poll measurement completed.
07:05:58.439 [INFO] {threadFn::rangeThreadFn}    Range read in 28.241 ms
07:05:58.439 [DBG]  {RangeSensor::readRange}    Starting range poll measurement...
07:05:58.443 [INFO] {camera::ExecGetPics}    Got a frame (1) in 5.058 ms to address 4415464
07:05:58.445 [INFO] {camera::ExecGetPics}    Got a frame (2) in 1.708 ms to address 4415736
07:05:58.445 [INFO] {camera::ExecGetPics}    Got both frames in 8.358 ms
07:05:58.445 [DBG]  {camera::GetPicsBlockingCapture}    Swapping cumulatve speed (from CAN
acceleration) tracking filters on tracker 1 and resetting the temp_filter...
07:05:58.445 [DBG]  {camera::CopyPicsToGPU}    Copying image 1 to GPU memory from 4415464...
07:05:58.447 [DBG]  {threadFn::outputThreadFn}    Output thread running for i = 0
07:05:58.447 [DBG]  {camera::CopyPicsToGPU}    Copying image 2 to GPU memory from 4415736...
07:05:58.448 [INFO] {threadFn::getPicsThreadFn}    [TIME] Got pics in 10.935 ms
07:05:58.448 [DBG]  {matching::getChange_px}    Offset X,Y: 0 x 0 | diff X,Y: 142 x 0
07:05:58.448 [DBG]  {matching::getChange_px}    Diff in array X,Y: 142 x 0
07:05:58.448 [DBG]  {threadFn::getPicsThreadFn}    [MATCH] Resetting calculations on speed tracker
1...
07:05:58.448 [DBG]  {threadFn::outputThreadFn}    [OUTPUT] Adding movement readings (X,Y: 142 x 0)
to speed tracker i: 0...
07:05:58.448 [DBG]  {matching::prepareForMatching}    pepareForMatching i: 1
07:05:58.448 [DBG]  {SpeedTracker::addReading}    Adding reading to SpeedTracker (movement X,Y: 142
x 0) 0...
07:05:58.448 [DBG]  {matching::prepareForMatching}    Reading template from first image...
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    Setting speeds from movement info...
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    axis 0, movementsum [mm]: 20.261895, t: 1988
07:05:58.448 [DBG]  {threadFn::getPicsThreadFn}    CURRENT_IMAGE_I toggled (now 0)
07:05:58.448 [DBG]  {matching::MatchingMethod}    MatchingMethod i: 1
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    v[i].mm_mcs: 0.010192
07:05:58.448 [DBG]  {matching::MatchingMethod}    Getting template search area...
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    axis 1, movementsum [mm]: 0.000000, t: 1988
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    v[i].mm_mcs: 0.000000
07:05:58.448 [INFO] {matching::MatchingMethod}    [TIME] Got template search area in 0.054 ms
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    axis 2, movementsum [mm]: 0.000000, t: 1988
07:05:58.448 [DBG]  {matching::MatchingMethod}    Starting template matching (src: 648 x 488,
templ: 250 x 150)...
07:05:58.448 [DBG]  {SpeedTracker::setSpeeds}    v[i].mm_mcs: 0.000000
07:05:58.448 [INFO] {threadFn::getPicsThreadFn}    [TIMING] getPicsThread sleeping for 2500 mcs...
07:05:58.448 [DBG]  {SpeedTracker::writeLog}    Writing tracker log (calculator: 1)...
07:05:58.448 [DBG]  {SpeedTracker::writeLog}    Writing "2019-07-03 07:05:58.448781, 0.00, 36.69,
0.00, 0.010192, 0.000000, 0.000000, 1, 20.261895, 0.000000, 1988, 0
"
07:05:58.448 [INFO] {can::execWriteToCan}    CAN TX:  0x350  [8]  0x27 0xD0 0x00 0x00 0x33 0x2B
0x5B 0x2D
```

2.3 Main function

Code 2.5. GSS main program main function in “main.cpp”

```
int main(int argc, char *argv[]) {
#if PROFILING
        nvtxNameOsThread(getpid(), "MAIN");
#endif

        printLog(DBG, "Main function begin; Starting initialization...");
        initialize(argc, argv);
        printLog(LOG, "Initialization completed. \n\n\n\n");


        if (OPTIONS.IMAGE_OUTPUT) { /// Window to show result
                printLog(DBG, "Creating image output window...");
                cv::namedWindow(image_window, cv::WINDOW_AUTOSIZE);
                cv::waitKey(1);
                printLog(DBG, "Image output window created.");
        }

        printLog(LOG, "Starting threads...");
        startThreads();
        printLog(INFO, "Threads started.");

        /// -- MAIN LOOP --
        printLog(DBG, "Changing STDOUT_LEVEL from %i to %i...", OPTIONS.STDOUT_LEVEL,
*OPTIONS.MAIN_LOOP_STDOUT_LEVEL);
        OPTIONS.STDOUT_LEVEL = *OPTIONS.MAIN_LOOP_STDOUT_LEVEL;

        while (!STOP_THREADS) {
                // show image (only works in main thread!) starting from 3. loop iteration ->
blocked for >10 ms!
                waitForSignal(SIG__END_MATCHING__START_SHOW);
                int show_I = !CURRENT_IMAGE_I;

                waitForSignal(SIG__OUTPUT_END__START_SHOW);
                if (MATCH_LOOP_I < 3) continue;

                if (OPTIONS.IMAGE_OUTPUT && ((MATCH_LOOP_I + OPTIONS.IMAGE_SHOW_RATE - 2) %
OPTIONS.IMAGE_SHOW_RATE == 0)) ShowInfo(show_I);
                //usleep(50000);

                if (OPTIONS.LOOP_LIMIT > 0 && MATCH_LOOP_I > OPTIONS.LOOP_LIMIT) {
                        printLog(DBG, "Loop limit %i reached (current loop: %i). Calling exit
handler...", OPTIONS.LOOP_LIMIT, MATCH_LOOP_I);
                        exit_handler(0);
                }
        }
        return 0;
}



void ShowInfo(bool i) {
        printLog(INFO, "Updating image in output window...");
        showMatchAreasOnImages(i);

        auto img = cv::Mat(imgs[i][1]);
        cv::imshow(image_window, img);
        printLog(DBG, "Output window updated.");

        printLog(DBG, "Waiting for keypresses on output window...");

        /// Check if "esc" key is pressed in image window -> end loop
        if ((cvWaitKey(3) & 255) == 27) exit_handler(1);
        printLog(DBG, "ESC was not pressed. Returning...");
}
```

2.4 Multithreading auxiliary code

Code 2.6. “threads.hpp”

```
#ifndef _THREADS_H
#define _THREADS_H

#include <vector>
#include <string>


#define waitForSignal(sig)    _waitForSignal(__FUNCTION__, sig, #sig); if
(__builtin_expect(STOP_THREADS, 0)) break
#define setSignal(sig)              _setSignal(__FUNCTION__, sig, #sig)
#define unsetSignal(sig)      _unsetSignal(__FUNCTION__, sig, #sig)

#define _SIG_COUNT 10
#define SIG__STARTED_MATCHING__START_GET_PICS 0
#define SIG__MATCHING_I_SAVED__TOGGLE_I 1
#define SIG__OUTPUT_I_SAVED__TOGGLE_I 5
#define SIG__END_PREPARE__START_MATCHING 2
#define SIG__END_MATCHING__START_OUTPUT 3
#define SIG__START_GET_PICS__START_ACC_TRACKING 4
#define SIG__OUTPUT_END__START_SHOW 6
#define SIG__END_MATCHING__START_GET_PICS 7
#define SIG__END_MATCHING__START_SHOW 8
#define SIG__END_SAVE__START_NEXT_CAPTURE 9


extern int RUNNING_THREADS;
extern std::vector<std::string> RUNNING_THREAD_LIST;
extern bool STOP_THREADS;


void _waitForSignal(const char* FUNC, int i, const char* sig_name);
void _setSignal(const char* FUNC, int i, const char* sig_name);
void _unsetSignal(const char* FUNC, int i, const char* sig_name, bool announce=true);

void startThreads();
void stopThreads();


#endif
```

Code 2.7. “threadFn.hpp”

```
#ifndef _THREADFN_H
#define _THREADFN_H


void getPicsThreadFn();
void matchThreadFn();
void outputThreadFn();
void rangeThreadFn(const int i);
void canReaderThreadFn();
void accResetThreadFn();


#endif
```

Code 2.8. “threads.cpp”

```
#include "threads.hpp"

#include <unistd.h>
#include <condition_variable>
#include <thread>
#include <pthread.h>
#include <boost/algorithm/string/join.hpp>

#include "threadFn.hpp"
#include "utils_time.hpp"
#include "log.hpp"
#include "options.hpp"

#if PROFILING
#include "/usr/local/cuda/include/nvToolsExt.h"
#include "/usr/local/cuda/include/cuda_profiler_api.h"
#endif


std::vector<std::string> RUNNING_THREAD_LIST;
bool STOP_THREADS = false;

std::thread rangeThread0, rangeThread1, outputThread, getPicsThread, canReaderThread, matchThread;


// { 0: match started, 1: match completed, 2: got pics, 3: CURRENT_IMAGE_I toggled }
bool SIGNAL_BOOLS[_SIG_COUNT];
std::condition_variable SIGNAL_VARS[_SIG_COUNT];
std::mutex SIGNAL_MTXS[_SIG_COUNT];

int RUNNING_THREADS = 0;



void _waitForSignal(const char* FUNC, int i, const char* sig_name) {
	auto t = getTime();
	if (OPTIONS.DBG_SIG) printLog(INFO, "[SIG] %s: Waiting for signal %i (%s)...", FUNC, i, sig_name);
	std::unique_lock<std::mutex> lock(SIGNAL_MTXS[i]);
	SIGNAL_VARS[i].wait(lock, [i]{return SIGNAL_BOOLS[i];});
	lock.unlock();
	_unsetSignal(FUNC, i, sig_name, false);
	if (OPTIONS.DBG_SIG || OPTIONS.DBG_SIG_WAIT) printLog(INFO, "[SIG] %s: After %.3f ms got (and removed) signal %i
(%s)", FUNC, getMs(t), i, sig_name);
}

void _setSignalValue(int i, bool val) {
	std::lock_guard<std::mutex> lock(SIGNAL_MTXS[i]);
	SIGNAL_BOOLS[i] = val;
}

void _setSignal(const char* FUNC, int i, const char* sig_name) {
	if (OPTIONS.DBG_SIG) printLog(INFO, "[SIG] %s: Setting signal %i (%s)", FUNC, i, sig_name);
	_setSignalValue(i, true);
	SIGNAL_VARS[i].notify_all();
}

void _unsetSignal(const char* FUNC, int i, const char* sig_name, bool announce) {
	if (announce && OPTIONS.DBG_SIG) printLog(INFO, "[SIG] %s: Removing signal %i (%s)", FUNC, i, sig_name);
	_setSignalValue(i, false);
}

void setThreadMaxPriority(pthread_t thId, uint offset=0) {
    pthread_attr_t thAttr;
    int policy = 0;
    int max_prio_for_policy = 0;

    pthread_attr_init(&thAttr);
    pthread_attr_getschedpolicy(&thAttr, &policy);
    max_prio_for_policy = sched_get_priority_max(policy);

    pthread_setschedprio(thId, max_prio_for_policy + offset);
    pthread_attr_destroy(&thAttr);
}
```

```
void setThreadAffinity(pthread_t thId, int cpu_i) {
        cpu_set_t cpuset;
    CPU_ZERO(&cpuset);
    CPU_SET(cpu_i, &cpuset);

    int rc = pthread_setaffinity_np(thId, sizeof(cpu_set_t), &cpuset);
    if (rc != 0) printLog(ERR, "Error calling pthread_setaffinity_np: %i: %s", rc, strerror(rc));
}

void startThreads() {
        if (RUNNING_THREADS > 0) {
                printf("ERROR: CANNOT START THREADS; THREADS ALREADY STARTED!\n");
        }

        if (OPTIONS.CAN_ENABLED > 1) canReaderThread = std::thread(canReaderThreadFn);
        if (OPTIONS.RANGE_ENABLED) {
                rangeThread0 = std::thread(rangeThreadFn, 0);
                rangeThread1 = std::thread(rangeThreadFn, 1);
        }
        outputThread = std::thread(outputThreadFn);
        matchThread = std::thread(matchThreadFn);
        getPicsThread = std::thread(getPicsThreadFn);
#if PROFILING
        nvtxNameOsThread(getPicsThread.native_handle(), "GetPics");
        nvtxNameOsThread(outputThread.native_handle(), "Output");
        nvtxNameOsThread(matchThread.native_handle(), "TemplateMatch");
#endif

        setThreadMaxPriority(pthread_self(), 10);

        setThreadMaxPriority(getPicsThread.native_handle());

        setThreadMaxPriority(matchThread.native_handle());
        setThreadAffinity(matchThread.native_handle(), 3);

        setThreadMaxPriority(outputThread.native_handle());

        printLog(DBG, "Threads started: %i", RUNNING_THREADS);
}

void stopThreads() {
#if PROFILING
        cudaProfilerStop();
#endif

        printLog(DBG, "Setting thread stop signal (for %i threads)...", RUNNING_THREADS);
        STOP_THREADS = true;
        printLog(INFO, "Thread stop signal set...");

        printLog(DBG, "Setting all signals while there are still threads running...");
        for (int count = 0; count < 5; count++) {
                printLog(DBG, "%i threads still running: %s", RUNNING_THREADS,
boost::algorithm::join(RUNNING_THREAD_LIST, ", ").c_str());

                for (int i = 0; i < _SIG_COUNT; i++) setSignal(i);

                usleep(1000000);

                if (RUNNING_THREADS == 0) break;
        }

        if (RUNNING_THREADS) {
                printLog(WARN, "Exiting the program with %i threads still running: %s", RUNNING_THREADS,
boost::algorithm::join(RUNNING_THREAD_LIST, ", ").c_str());
        } else {
                printLog(INFO, "All threads have stopped.");
        }
}
```

2.5 Camera integration

Code 2.9. “camera.hpp”

```
#ifndef _CAMERA_H
#define _CAMERA_H

#include <m3api/xiApi.h>


// resolution 648x488
#define X_RES (648)
#define Y_RES (488)

/// Define camera parameters
// Timeout replaced by OPTIONS.MAX_DELAY
#define AEAG false          // auto exposure, auto gain


/// Ximea Global Variables
extern XI_IMG xiImgs[2][2];


// acquired image utility functions
unsigned int getImgDelay(XI_IMG img1, XI_IMG img2);

// image acquisition functions
void GetPicsBlockingCapture(int threadI);

// camera utility functions
void SetUpCamera();
bool CameraConnected();
void ShutDownCamera();


#endif
```