Universidad Carlos III de Madrid

Trabajo de fin de estudios

Grado en Ingeniería Electrónica Industrial y Automática

2017-2018

THERMAL VISION-BASED MAPPING FOR UNMANNED AERIAL VEHICLES

Andrés Prieto Yanes

Abdulla Al-Kaff

June 2018, Leganés

Abstract

Autonomous Mobile Robots investment is growing. Consequently, they need to percept the environment in many different conditions. Thermal cameras bring the possibility to fill the gap in conditions that ordinary vision sensor leave. In this thesis, an adaptation of the state of the art Visual SLAM software, ORB-SLAM 2, for thermal cameras is presented. This implementation is presented alongside an example application with a Stereo camera based on an easily expandable MIMO system.

Acknowledgements

To my mentor and friend, Eduardo Martín Blecua, for always believing in me when I didn't and guiding me to find the solution by myself.

CHAPTER 1

Introduction

"...robotics navigation, where a robot must estimate its location relative to its environment, while simultaneously avoiding any dangerous obstacles." Richard *Szeliski*

1.1 INTRODUCTION

Thanks to the computing development over the last years, real time perception systems have come to reality. These perception algorithms can tell a robot how is the environment that surrounds it and at the same time solve for the position of the robot in that environment by performing what is called SLAM (Simultaneous Localization and Mapping). The robot must know the environment to achieve a goal successfully, safely for itself and for the environment (people, objects or facilities).

1.2 PROBLEM STATEMENT

Aerial vehicles need to localize themselves to perform diverse tasks. A mobile robot normally fuses data from different sensors (e.g encoders that read the angle of the wheel) and given certain physical restrictions (e.g. the robot moves in a 2D plane and there is no sliding in the wheels). Aerial vehicles do not have a reliable and fast way to localize themselves as ground robots do, that is why visual odometry is normally required. This visual odometry can suffer under different illumination conditions such as dusk, smoke or low luminance.

Autonomous Mobile Robots' control architecture have normally a deliberate and a reactive layer. The robot needs to react fast to changes in the map, those changes will occur specially in outdoor applications where the environment can be unknown in the throughout the mission. In this sense the robot has to have a high frequency information channel for the information on the environment. GNSS (Global Navigation Satellite System) can give an absolute position but the frequency and delays have to be taken into account. Those restrictions make additional inputs to the estimation necessary.

Having an UAV patrolling a place can give cheap and valuable information compared to the cost of having a helicopter doing the same route. It also gives the possibility to do routes that are not possible using traditional air vehicles. This routes are often done in poor visibility. Thermal cameras perform better than conventional cameras in low visibility conditions[15]. As an example, flying an UAV with a thermal-based SLAM system over last summer Portugal's fire could have given relevant input for the fire-fighters. Certainly, using a multi-spectral based SLAM (Visual and thermal spectrum at the same time) can palliate the problems that separately could be present. Furthermore, in rescue applications thermal cameras can be useful to detect a victim [6]

1.3 PROPOSED SOLUTION

The proposed solution is to use a thermal camera alongside a RGB Stereo camera and the regular sensors that normally are on-board the UAV (Inertial Measurement Unit, GNSS) for SLAM. Given that we have at least the thermal camera and the stereo camera to get a position estimate of the robot. Moreover, since different applications normally imply different sensors and actuators, the estimator has to be able to handle different inputs with minor changes in the code.

But why using a Thermal camera? Conventional cameras make SLAM algorithm to suffer under sudden illumination changes[5]. Moreover, thermal cameras are decreasing in price and the advantages that can provide are starting to worth the cost in certain applications rescue and terrain analysis.

1.4 REGULATORY FRAMEWORK

The regulations that must be taken into account on this project vary according to the application and the country in which is used. I would like to focus more on the aerial vehicles since those can be the most restricted ones. On this project, the bare fact that a thermal camera was used does not mean any challenge compared to the fact that a Drone is used.

1.4.1 *Indoor applications*

On indoor applications, aerial space regulations are not applied directly since the aerial space does not belong to the State. Thus, just an operator and the authorisation of the property would be needed. Both assuming the risk associated to the application given by a proper security study.

SORA (Specific Operations Risk Assessment) from JARUS is normally advised as a methodology to use; failing this, Scandinavian methodology, used by most of airlines could be used.

1.4.2 *Outdoor applications*

The current Spanish regulation applied in Spain is RD 1036/2017 published December 29th 2017. The current regulation does not allow fully autonomous flight, just automatic given that the pilot cannot loose control of the Drone. This type of flight is called BVLOS (Beyond Visual Line of Sight) and an advanced pilot certification is required plus a NOTAM (Notice To Airmen) with its associated TSA.

This Spanish regulation is not part of the EASA (European Aviation Safety Agency) regulations. This implies that this regulation does not apply to the rest of the European Union. Therefore, each State has its own regulations. Although, there is a group of experts gathered to state guidelines on this topic called JARUS (Joint Authorities for Rulemaking on Unmanned Systems). This group works on giving a "single set of technical, safety and operational requirements for the certification and the safe integration of Unmanned Aircraft Systems (UAS) in airspace and at aerodrome" [9]

In CS-LURS V.1.0 page 111 it states that: *"Electrical connections of externally mounted payload and accessories, such as cameras, should be sufficiently protected to preclude electrical fires and the devices should not be likely to penetrate a fuel compartment."* This is a valuable point, given the application in an aerial vehicle, could be possible to power the thermal camera with the same source of electricity as the vehicle to minimize weight. That connection needs to be done in a safe way so that there are no undesired shortcuts that would endanger the mission.

Contents

CHAPTER 2

State of the art

2.1 Application

As introduced above the goal in this project is to use Thermal cameras as an input to Visual SLAM for Mobile Robots. The measurements are taken alongside other common sensors in mobile robots such as LIDAR, IMU and other visual spectrum cameras.

LIDAR is a widely used sensor in Mobile robotics. The most common technology relies on one or several laser distance meters that take measurements on a circular movement. This sensor has good accuracy and can give high volume of information that can be analysed conforming a relatively dense point cloud comparing to Visual methods. A good SLAM can be obtained just by using this sensor. Although, weather can heavily harm this measurements.

IMUs are present in most of AMR. This subsystem outputs measurements of angle and acceleration at high frequency. By integration and differentiation we can obtain velocities, position, angular speeds and angular accelerations. The problem relies on the vias that these sensors have (specially the affordable ones).

On the next page you will find table 2.1 in which the assumed properties of the sensor are stated. We can clearly see that each sensor has its strengths and weaknesses. This is why multi-sensor systems are made; a robot has to be able to work safely for it, for the people and for the environment that surrounds it. Specially in outdoor applications several conditions can make the perception to suffer. Weather conditions might obscure LIDAR measurements, smoke can be present in the environment harming RGB camera sight...

Table 2.1 – Application assumed properties of the sensors.

Sensor		**LIDAR**	**IMU**	**Wheel encoder**	**GNSS**	**RGB camera**	**Thermal camera**
Robustness to	**Rain, Dust...**	Very low	-	-	-	Very low	High
	Low illuminance	High	-	-	-	Low	High
	High illuminance	High	-	-	-	Low	High
Data interruption and delays		Unusual	Unusual	Unusual	Present	Unusual	Present
Frequency		High	High	High	Very Low	Moderate	Moderate
Vias and/or tolerances impact		Low	High	High	Low	Low	Low
Price		€€€	€	€	€	€	€€
Resolution, accuracy aspects		High accuracy	Low accuracy	High accuracy	Low accuracy	High resolution	Low resolution

As previously discussed, aerial information can be taken from a multirotor at a lower cost compared to conventional methods such as Helicopters. This opens the possibility to get information on rescue and prevention missions. Since these cameras can offer a great benefit even before applying AI (Just by visual inspection) we believed it was a great opportunity to research on the benefits that these cameras can provide for robot localization. DJI, which is one of the biggest companies in drones and FLIR, which is also well-known for their infrared cameras are investigating this opportunity and are developing a drone with infrared vision. The applications mentioned in their new non cooled infrared camera ZENMUSE XT are promising.

- Grid inspection
- Solar Panel inspection
- Search and rescue
- Agriculture crop monitoring
- Fire fighting

Indoor localization is also growing in mobile robotics. These applications can be both on a terrestrial or aerial settings. The mining industry is making a big investment on mobile robots and specially on poor visibility conditions which make thermal or infrared vision a good ally.

KAIST In autonomous cars field, KAIST (Korea Advanced Institute of Science and Technology) developed *Multispectral Pedestrian Dataset* (color + infrared) and used it to take video of a walk-through by car of a city. Tracking or distinguishing pedestrians is supposed to be easier on infrared due to the high temperature of our bodies. This is a great possibility for colission evasion, specially at night where even the human eye could not evade a collision.

Field robots Mining The mining industry is starting to use more AMR equipped with thermal cameras. "In a mining environment, if visible light or digital cameras are used for monitoring, they will miss out on any potential defects as mining environments are often very dusty. Thermal can see through dust and smoke due to their infrared wavelength, meaning that it is able to detect any heat energy through most environmental conditions." [4]

2.2 RELATED WORKS

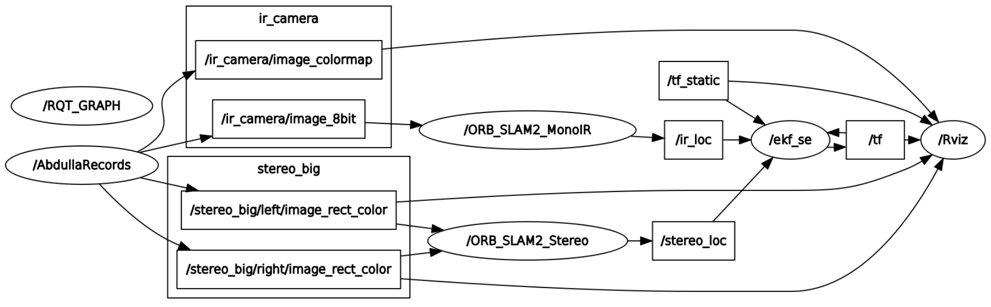
Thermalvis by Stephen Vidas is a ROS package that tackled this problem some years ago. "It was motivated by the need to adapt several computer vision methods to thermal-infrared video. In particular, methods relating to camera calibration, local feature tracking and 3D motion estimation. However, the flexibility required to achieve this has resulted in several methods and interfaces that should also be useful for people using regular video, or video from other spectral ranges." says Vidas [18]. His work was really valuable for this project to understand the possibilities and limitations of Thermal cameras. As an example, in Thermalvis he deals with the data interruption in the frames by detecting it (NUC) and using that time to gather more information from the last published frame.

CHAPTER 3

Proposed Algorithm

3.1 Mobile Robot Localization

The project is intended to give a localization given several inputs. In the implementation data from a stereo camera and a thermal camera are used. Each source of information is processed by a minimally edited ORB-SLAM 2 version independently and Robot Localization performs pose estimation from both inputs. This pose estimation is done by applying an Extended Kalman Filter (EKF) of the inputs. The whole system is connected by ROS so that more sensors can be added with no mayor changes in the code as thought in the proposed solution. The code can be found in the repository under request.



The OS system used was Ubuntu 16 64 bits with ROS Kinetic. This set was used since most of the community of Autonomous Mobile Robots are using Ubuntu and it is normally the operative system that the on-board hardware has. For example Beaglebone, Raspberry Pi and Odroid can use this OS.

Furthermore ORB SLAM 2 troubleshooting is big on Ubuntu. C++ was used because it appears to be more optimal than Python and also since ORB SLAM is coded in this language.

Before explaining what is a thermal and stereo video camera I would like to briefly introduce you how conventional digital cameras work. A digital picture can be seen as a group of small elements called **pixels**. Each pixel has a level of light intensity associated. Although, how can we capture colours then? We can define every colour as a combination of red, green and blue.

This conventional cameras show us pictures from the visible spectrum (400-700 nanometers) but thermal cameras or infrared cameras take pictures from $7.5 - 13\mu$m, which mean that we can see heat. This pictures only have one channel (grayscale that ranges from 0% as black and 100% as white).

Video is the acquisition of pictures within a small time lapse between them. These pictures are called frames and the speed in which you take frames is expressed in frames per second (fps). In the experiments we get video from a thermal camera and a stereo camera.

3.2 CAMERA CALIBRATION

THE IDEA OF OBTAINING a 3D point from a monocular camera (2D) is something that at first sounds like cheating. Although, if we know the camera parameters we can know the ray (represented in red in figure 3.1) that is composed by all of the possible 3D points that have a projection. Moreover, if we have several images and we know the pixel correspondences between them, we can determine the position of that points and track how is camera frame moving.

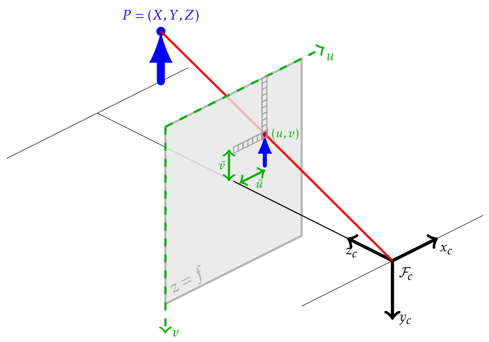


FIGURE 3.1 – Perspective representation (Edited version of [1]).

3.2.1 *Camera Intrinsic Matrix*

Even though camera manufacture is really precise nowadays, every camera has unique intrinsic parameters:

$$K = \begin{pmatrix} f_x & s & x_o \\ 0 & f_y & y_o \\ 0 & 0 & 1 \end{pmatrix}$$

3.2.1.1 f_x and f_y:

This parameters are the ratio between the focal length and the size of the pixel in x and y directions respectively. Both parameters can be considered equal and the results would not be affected with huge error. In other words, we would assume perfectly squared pixels.

3.2.2 *Skew (s):*

This parameter represents how non perpendicular are the sensor axis. The closer **s** is to 0 the more perpendicular the axis are.

3.2.2.1 x_o and y_o

This parameters are the coordinates of the center of the camera plane. One might think that is the number of pixels in each direction divided by two but, again, due to manufacturing imprecisions it is normally decimals away from that ideal center.

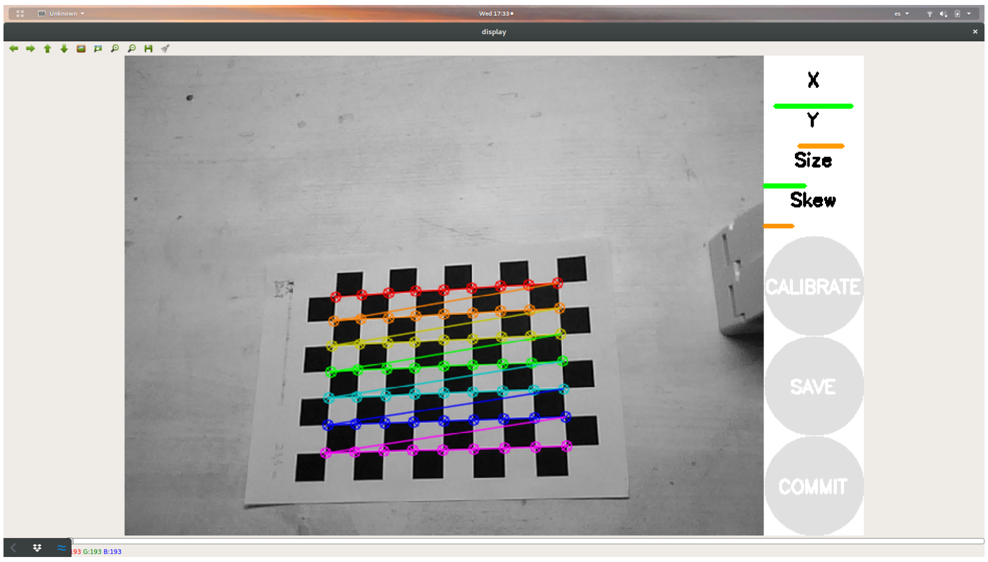
3.2.3 *Obtaining the parameters*

The most popular and simplest way to obtain the camera parameters is to use a known image placed in a scene and take several pictures of that image. Typically a chequerboard is used and by giving the length of each square and the number of squares in the chequerboard we can solve for the camera parameters.

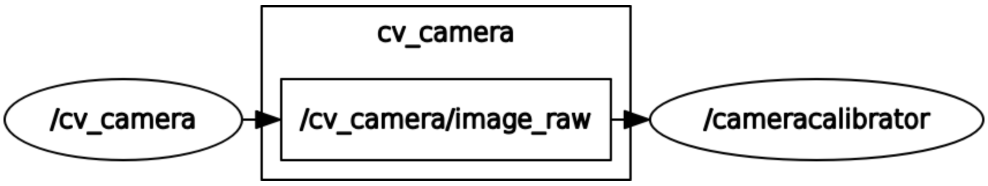
I used the *camera calibration* package from ROS to live calibrate the camera. The procedure is simple:

- We publish pictures from the camera to a topic.
- We change the position and orientation of the chequerboard in the scene.
- The program will tell us when the data is sufficient
- Finally it outputs a yaml file with the parameters

ORB SLAM uses a yaml file to specify the camera calibration parameters. I used ROS camera_calibration package to obtain these parameters specifying the node for the image acquisition.



```
rosrun camera_calibration cameracalibrator.py --size 7x9
--square 0.235
image:=/cv_camera/image_rawcamera:=/cv_camera
```



Then the program exports the images and the camera parameters. Unfortunately the format is not the same as the one ORB SLAM 2 uses but we can just fill the yaml file with the obtained camera parameters.

3.3 STEREO CAMERA

As humans we can see depth by comparing two images from our eyes. The stereo camera uses this principle. Two cameras displayed normally coplanar and arranged at a certain distance from each other called *baseline*.

Comparing these two images, in practice, means that we detect points from one image and look for an alike point in the other. These points have to be unique so that we don't mismatch them. To make this possible the idea of corner was developed thanks to the work of Chris Harris and Mike Stephens [8]. These corners are regions in the image in which the gradient changes abruptly in two linearly independent directions.

Once we found the corners of an image we need to describe them so that later we can compare them to another corner in another image. To do this descriptors are used. A descriptor is a tool used to describe a certain point in an image (e.g. FAST, SURF...) but with scale and rotation invariance. It is important due to the fact that the corner can be rotated or at a different size in the other image.

Looking for this descriptors can be done efficiently by applying SSD (Sum of Square differences) or Cross Correlation and the epipolar line constraint. This epipolar constraint is based on the assumption that the point detected in one camera can only land on a line in the other camera frame. In this sense the algorithm searches for this feature along the line. Epipolar lines are horizontal in our case since both frames have the same pose but shifted by the baseline.

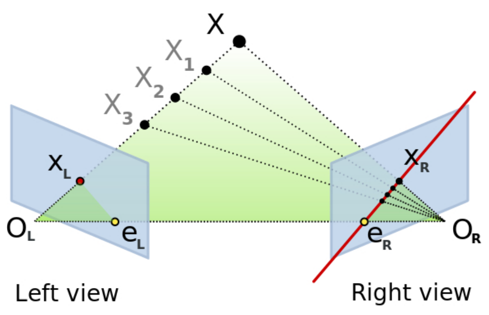


FIGURE 3.2 – Epipolar geometry.

Once matching is completed we can compute the depth of a point

$$\tan(\theta) = \frac{f}{x_r} = \frac{Z_p}{X_p} = \frac{Z_p}{X_p - b}$$

$$\frac{f}{x_r} = \frac{Z_p}{X_p - b}$$

By computing the tangent of both triangles we obtain those equations. And now we can try to solve for Z_p with the data given.

$$\begin{aligned} f(X_p - b) &= Z_p x_r \\ f X_p &= x_1 Z_p \end{aligned}$$

$$-f b = Z_p(x_r - x_1) = Z_p(-d)$$

$$Z_p = \frac{fb}{d}$$

Where $x_r - x_l$ is defined as disparity.

3.4 THERMAL CAMERA

As introduced above, thermal cameras can capture pictures in the Infrared spectrum of light. But using thermal cameras present several limitations[7, 18]:

- Lower resolution
- Poor SNR (Signal-Noise Ratio)
- Data interruptions

The NU noise problem One of the constrains of thermal cameras is data interruption. Thermal camera pixels have a non-uniform (NU) response in terms of operation point and sensitivities[3]. "Non-uniformity can be considered as a 1D flicker of the columns inside each frame"[16]. This visually translates in lines in rows or columns (good examples can be found in the article from Tendero, Landeau, and Gilles).

To deal with this problem thermal camera performs the *non-uniformity correction* that produces the camera to freeze in a frame for even more than a second. This delay can harm the tracking, specially if the NUC is happening while moving[18]

3.5 ORB SLAM 2:

ORB-SLAM 2 is the second iteration of "*A versatile and Accurate Monocular SLAM System*"[10]. This software computes the trajectory of the camera frame in real time.

This software uses ORB descriptor which is developed based on FAST keypoint detector and BRIEF descriptor [12]. The keypoint detector can be tuned with a threshold given in the camera parameter file which is advised to lower it when a low resolution image is used. It is important for our application considering that the Thermal camera has a resolution of 164x129 (which is low).

The developers of the software provide a Vocabulary so that the system is already trained to recognize features from the application environment. In this case the system is trained for urban ground footage but the system can be trained to recognize features from an aerial point of view and even to get vocabulary for thermal view. For testing purposes I did not create a specific vocabulary for the application.

This software has presented good results with Monocular, Stereo and RGB-D cameras on a CPU based system and even better results in the GPU implementation. Although, this does not guarantee that this implementation would work with the limitations that thermal cameras present. The system implemented works in a ROS environment. Although, ORB-SLAM 2 is not fully implemented to work on it and some modifications had to be made for the application.

3.6 LAUNCH

The system can be launched by coding a file in which we state all the connections between nodes of the system as shown in figure 3.1 To launch all the nodes and connections I wrote this launch file:

```
<launch>
<node name="ORB_SLAM2_Stereo" pkg="ORB_SLAM2" type="Stereo"
args="path_to_calibraton  false">
<remap from="/stereo/left" to=
"/stereo_big/left/image_rect_color"/>
<remap from="/stereo/right" to="/stereo_big/right/image_
rect_color"/>
</node>
```

The first node to be launched is ORB SLAM 2 Stereo which takes frames from two topics and performs the calculation of the trajectory. The default names of the topics that ORB SLAM 2 subscribes to don't match with the ones in which we are publishing. I used the remap command to change the name at which the node will subscribe to.

```
<node name="ORB_SLAM2_MonoIR" pkg="ORB_SLAM2"
type="MonoIR" args="path to calibration ir false">
<remap from="/image" to="/ir_camera/image_8bit"/>
</node>
```

Secondly, the Thermal camera slam node is launched and conveniently remapped.

```
<node pkg="rosbag" type="play" args="path to bagfiles"/>
```

Our source of data is in our case a rosbag from a mobile robot so we play the recorded data from a footage around the university.

```
<node pkg="robot_localization" type="ekf_localization_node"
name="ekf_se" clear_params="true">
    <rosparam command="load" file="ekf parameters" />
</launch>
```

Finally Robot localization is launched.

3.7 MONOCULAR IR NODE

In this node I modified ORB-SLAM 2 so that it would have an extended point object. This point would not only have a three coordinate vector but also would have a value for the temperature on that neighbourhood. We can know that temperature by reading the value of that pixel from the current frame higher values correspond to higher temperatures, that means the whiter the pixel the higher the temperature on that point. In theory, the 3D point must have the temperature represented in the pixel since we know the camera calibration.

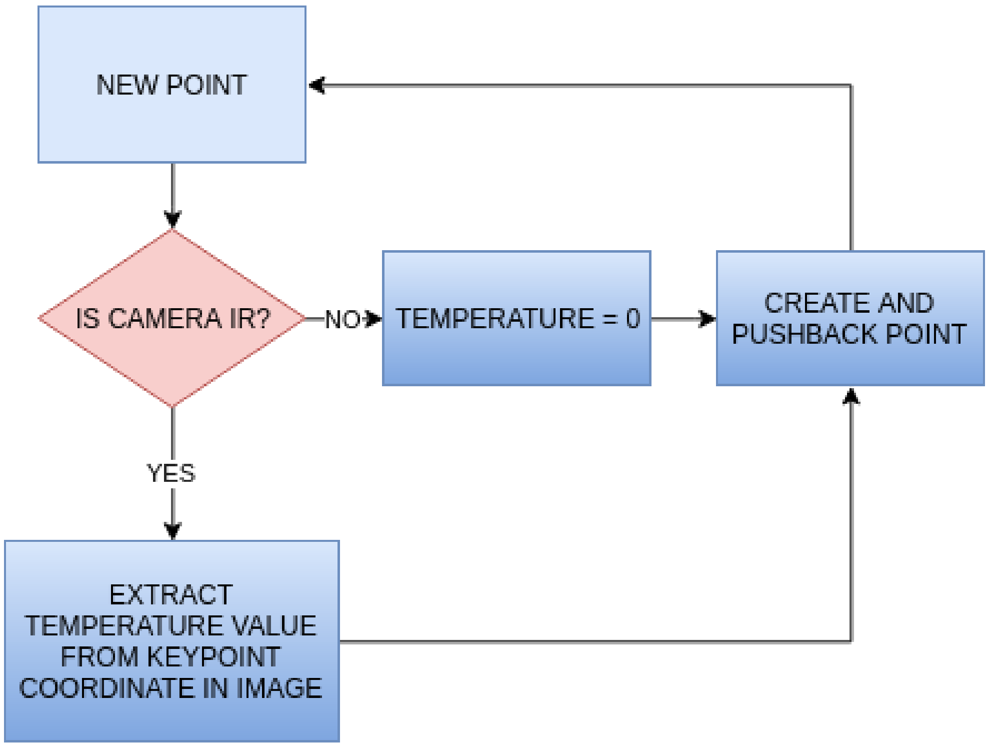

NEW POINT
IS CAMERA IR?
NO
TEMPERATURE = 0
CREATE AND PUSHBACK POINT
YES
EXTRACT TEMPERATURE VALUE FROM KEYPOINT COORDINATE IN IMAGE

Figure 3.3 – New Map Point.

Figure 3.3 explains how the implementation assigns temperatures. When a new point is proposed to be created by the Tracking node, the algorithm checks the information from the camera provided by *cameraCalibration.yaml* and if the camera is infrared we can get the data of the temperature of that point from the current image frame and then create the extended MapPoint object. In case of a non-ir camera the system would just assign a value of 0 in the temperature field. This null value is treated by the algorithm as no temperature on that point.

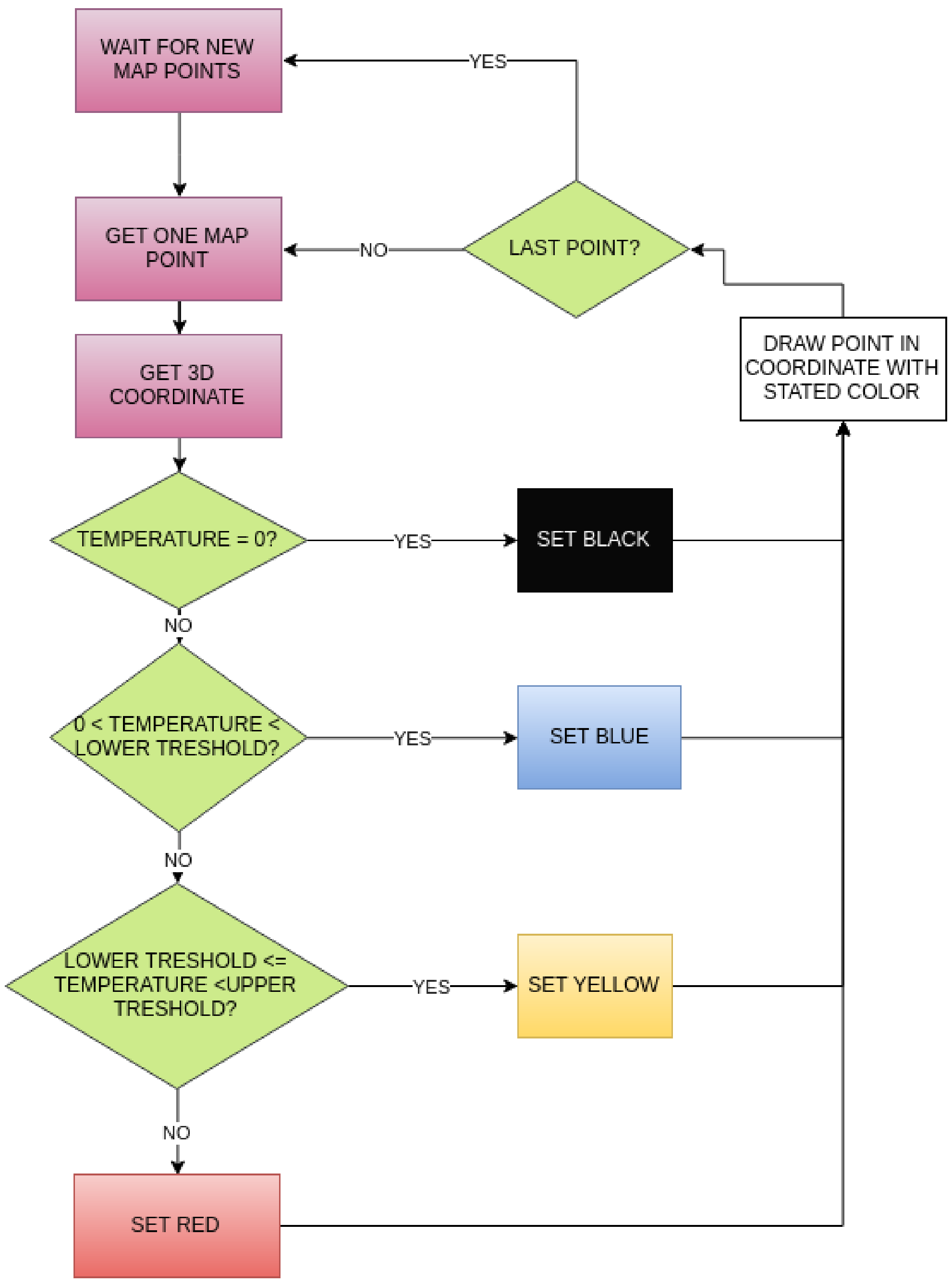


FIGURE 3.4 – Draw Map Point.

This part of the algorithm, represented in figure 3.4, maps the temperature from the attribute on each point and draws it in the 3D map according to that mapping. A quite discrete mapping rule was used in the implementation. It assigns black to non ir points and depending on the temperature treshold it colours the points in blue, yellow and red. A more gradual mapping of temperature was also implemented. This mapping was done with OpenCV applyColorMap function with a JET mapping. This mapping is a more visual and familiar way to see temperature since the camera only gives you a range of temperature in greyscale. This mapping can be seen in the following picture:

COLORMAP_JET

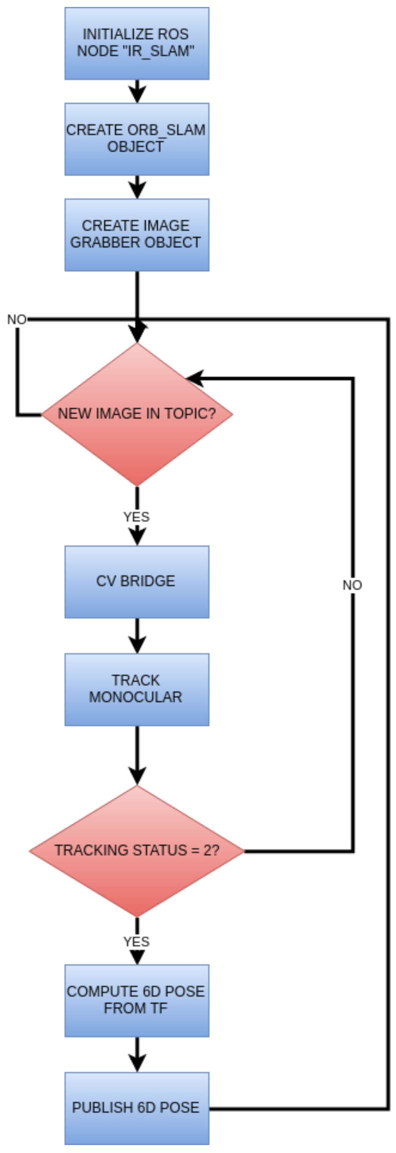
Figure 3.5 – Temperature Mapping.

3.7.1 *Using ORB-SLAM 2 alongside ROS*

In this section the most important part of the code that launches ORB SLAM 2 for the Thermal Camera will be explained.

ORB SLAM is not integrated in ROS but we can feed the system with images from a ROS topic and publish the pose that the system outputs on a topic. The program first initializes the node on ROS with the name IR_SLAM. Then it creates an ORB SLAM system object. It needs a path to the Vocabulary and a boolean to decide if we want to visualize the system as it runs. This is convenient to reduce CPU workload when the viewer is disabled. Furthermore, we create an Image Grabber object that is in charge of feeding images to the system. It need the reference to the system.

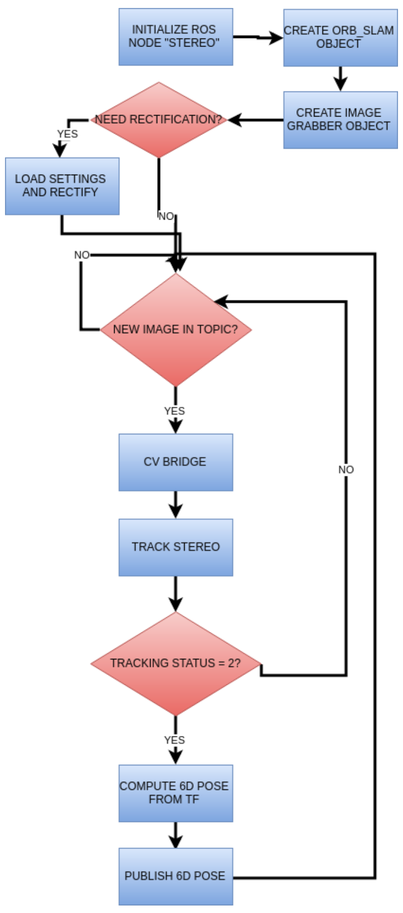
Later the system waits until new frames are published into the topic. When a new image is published cv_bridge processes the message. *cv_bridge* is a crucial package for the program since it gets ROS image messages and converts it into OpenCV Mat type; which is the data type that ORB SLAM 2 uses. In other words works as a translator between ROS and ORB SLAM 2. When the image is converted to OpenCV format the system is fed with it. ORB SLAM 2 needs to be initialized, this means that the system has to solve for a good set of points to create a



sufficiently good map in the beginning from which it will start expanding the map from. The system status can be obtained so that if the SLAM is not initialized the node will not publish any message for pose. If the system is initialized (Status = 2) the node will solve for the 6D pose given the transformation of frame matrix from Track Monocular Function.

3.8 STEREO

THE STEREO NODE is quite similar to the monocular thermal node. As explained in the camera calibration section, to optimize the search of features to a one dimensional problem we do rectification if needed. In our experiments we did not need that rectification since the camera has already two coplanar frames. For stereo rectification the system needs the camera parameters. These camera parameters can be found in the yaml file provided at initialization. To initialize the system we do the same procedure as in the monocular case but we tell ORB SLAM 2 that the type of tracking will be stereo. In the case of Stereo we take 2 images each loop.



EKF

The Extended Kalman Filter is a suboptimal filter for non-linear systems on a stochastic model base. Since the optimal estimator for discrete non-linear systems is difficult to implement (Due to high requirements on memory and computation) I used the EKF. This explanation is based on the book by Bar-Shalom [2].

3.9 Modelling

3.9.1 *State space model*

The general model for a stochastic model in discrete time:

$$x(k+1) = \mathbf{f}[k, x(k), u(k), v(k)] \tag{3.1}$$

Where x is the space vector, u is the input vector and v is the process noise. Generally, **f** function is time variant. For this model the noise is considered white, zero mean (expectation is zero) and additive so that the model is:

$$x(k+1) = f[k, x(k), u(k)] + v(k) \tag{3.2}$$

$$E[v(k)] = 0 \tag{3.3}$$

$$E[v(k)v(j)'] = Q(k)\delta_{kj} \tag{3.4}$$

3.9.2 *Measurement model*

Measurements are derived from the general model:

$$z(k) = \mathbf{h}[k, x(k), w(k)] \tag{3.5}$$

Where w is measurement noise. We will consider it also white, zero mean and additive.

$$z(k) = h[k, x(k)] + w(k) \tag{3.6}$$

$$E[w(k)] = 0 \tag{3.7}$$

$$E[w(k)w(j)'] = R(k)\delta_{kj} \tag{3.8}$$

We will suppose also that noises are not correlated.

3.10 ONE LOOP ON THE ALGORITHM:

A first order EKF is based on a first order Taylor series expansion of the non linear functions. In the dynamics and, if it was the case, in measurement equation. The second point is to use the LMMSE (Linear minimum mean square error estimator)

The algorithm can be understood better with this diagram [19]:

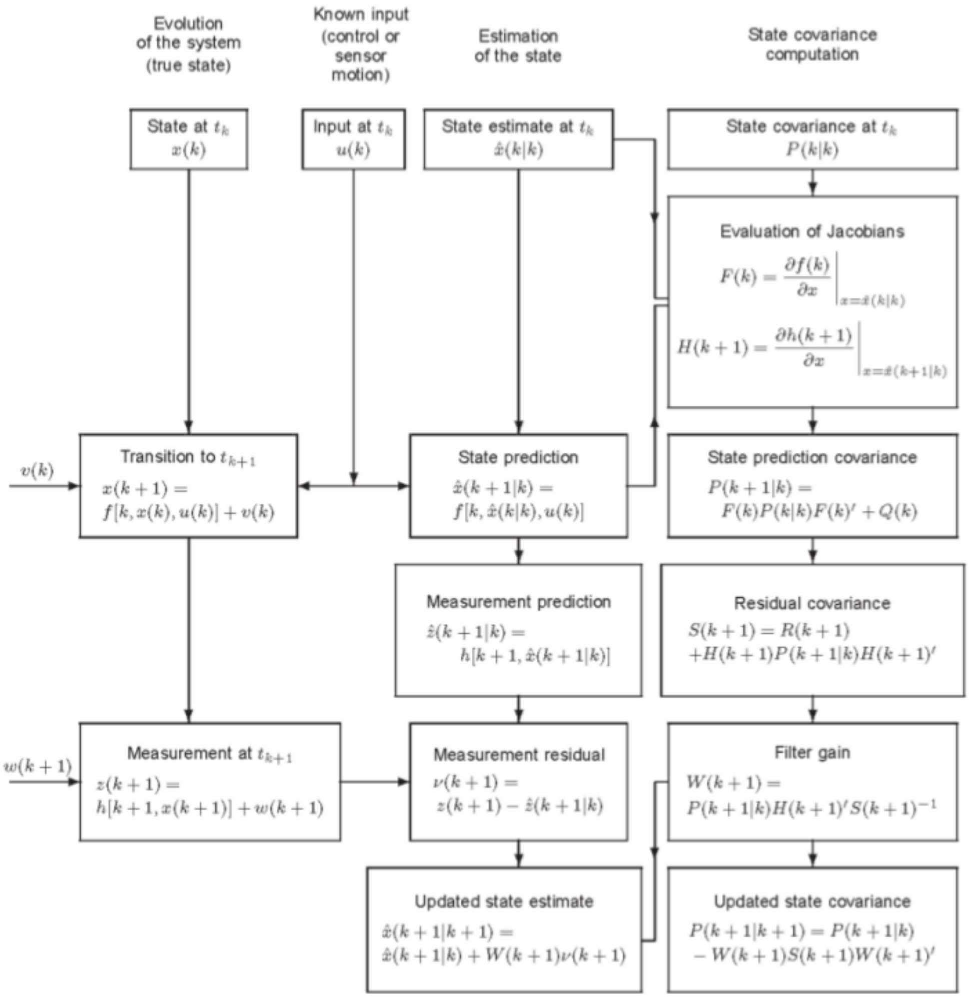

Evolution of the system (true state)
Known input (control or sensor motion)
Estimation of the state
State covariance computation
State at t_k $x(k)$
Input at t_k $u(k)$
State estimate at t_k $\hat{x}(k|k)$
State covariance at t_k $P(k|k)$
Evaluation of Jacobians
$F(k) = \left.\frac{\partial f(k)}{\partial x}\right|_{x=\hat{x}(k|k)}$
$H(k+1) = \left.\frac{\partial h(k+1)}{\partial x}\right|_{x=\hat{x}(k+1|k)}$
$v(k)$
Transition to t_{k+1} $x(k+1) = f[k, x(k), u(k)] + v(k)$
State prediction $\hat{x}(k+1|k) = f[k, \hat{x}(k|k), u(k)]$
State prediction covariance $P(k+1|k) = F(k)P(k|k)F(k)' + Q(k)$
Measurement prediction $\hat{z}(k+1|k) = h[k+1, \hat{x}(k+1|k)]$
Residual covariance $S(k+1) = R(k+1) + H(k+1)P(k+1|k)H(k+1)'$
$w(k+1)$
Measurement at t_{k+1} $z(k+1) = h[k+1, x(k+1)] + w(k+1)$
Measurement residual $\nu(k+1) = z(k+1) - \hat{z}(k+1|k)$
Filter gain $W(k+1) = P(k+1|k)H(k+1)'S(k+1)^{-1}$
Updated state estimate $\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + W(k+1)\nu(k+1)$
Updated state covariance $P(k+1|k+1) = P(k+1|k) - W(k+1)S(k+1)W(k+1)'$

FIGURE 3.6 – EKF.

3.10.1 *Initial Value or previous estimate*

We first receive a previous estimate and the covariance matrix. With these values we have all the information needed for the estimator due to Markov property.

3.10.2 *State prediction*

We can use the system dynamics equation to obtain the a priori estimate for $k+1$ given the data until k. This can be expressed as $\hat{x}(k+1|k)$.

3.10.3 *Jacobian evaluation*

We linearise f and h functions around the point of operation. And here we can see that the state space function does not have to be invariant in time necessarily.

3.10.4 *State covariance matrix update*

We calculate the new a priori state covariance matrix $P(k+1|k)$. It is interesting to say that Q increases our covariance each loop, meaning more uncertainty from that input.

3.10.5 *Residual covariance*

At this point we use the new H and also add the measurement covariance R.

3.10.6 *Kalman gain*

This gain weights the importance we give to the estimation or measurement depending on the uncertainty.

3.10.7 *A posteriori state covariance update*

It is important to point out the minus sign in the second term, it tells us that the uncertainty is getting smaller so that our estimate is getting more precise.

3.10.8 *A posteriori measurement prediction*

In this step we calculate an estimate for the next measurement in which we use the a priori state estimate $\hat{x}(k+1|k)$.

3.10.9 *Innovation vector*

Innovation vector will be multiplied by the Kalman Gain to correct the estimate with the a posteriori data. This vector can detect system failure. If the vector follows a white noise, zero mean sequence the estimator is working fine.

Robot Localization

Once the algorithm was selected, the next step was apply it. For this purpose I used Robot Localization ROS package. This package allows you to estimate the pose of the robot on a 6 dimensions frame by applying the Extended Kalman Filter explained above.

Using this package with ROS offers the possibility to add inputs to the estimate (not covered on the project) from MAVROS. MAVROS is a package from ROS that handles information from the most common civil drones software. This node can input GPS signal and IMU mainly for the estimation. The scope of this project does not contemplate the inclusion of these inputs but using a flexible package gives better possibilities in future implementations.

3.11 setup robot localization

Robot localization needs some parameters to filter the data that can be loaded from a file, this file can be found on *params* with the name *ekf_template.yaml*. In this case two inputs were used: the output from the stereo camera and the output from the thermal camera. No probabilistic model was used for the output data from ORB SLAM.

3.11.1 *Filter frequency*

The frequency for the filter was left the same as in the template (30 Hz) since both nodes are giving poses at a lower frequency than that 30 Hz, if the fps were higher, for example by using a different camera this parameter should be increased. The same concept applies if an IMU or LIDAR was added to the estimation.

3.11.2 *Odometry configuration matrix*

The "Each sensor reading updates some or all of the filter's state. These options give you greater control over which values from each measurement are fed to the filter." The node needs to know which of the measurements will be received. For example there might be an accelerometer that gives data on acceleration on x and z axis, in this sense the node needs to know which data will be received. To specify it a configuration matrix was defined in a similar was to the C matrix in state space representation; a 1 means that the input will be provided.

$$odomConfig = \begin{pmatrix} x & y & z \\ \phi & \Theta & \psi \\ \dot{x} & \dot{y} & \dot{z} \\ \dot{\phi} & \dot{\Theta} & \dot{\psi} \\ \ddot{x} & \ddot{y} & \ddot{z} \end{pmatrix}$$

For both inputs the matrix is:

$$odomConfig = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Which means that we are receiving angular and linear position.

CHAPTER 4

Experimental Results

4.1 KAIST MATERIAL

I downloaded *KAIST Multispectral Pedestrian Detection Benchmark* dataset as a test for the monocular node with a thermal camera information. The program was ran through terminal given the paths to the vocabulary and the guessed calibration file.

```
andres@Z50:~$ rosrun ORB_SLAM2 Mono
Vocabulary/ORBvoc.txt calibration/FLIRKAIST.yaml
```

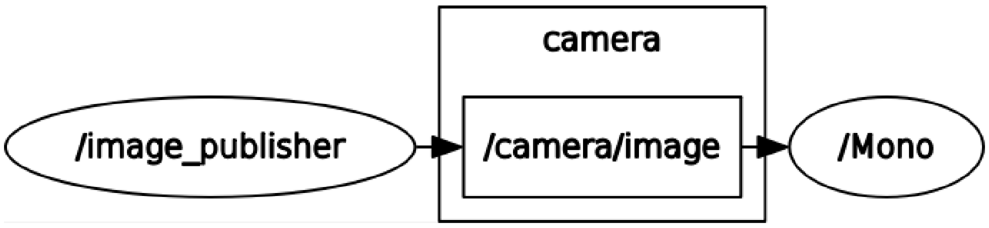


Figure 4.1 – Connection between camera and ORB-SLAM 2.

This project uses a multispectral camera (RGB and a Thermal FLIR aligned cameras) for pedestrian detection both in daytime and night-time. This is valuable to test how the thermal camera responds to different conditions. The camera calibration was not provided by the dataset and that could affect the tests. Even though the results are promising despite the fact that a proper camera calibration file would probably improve the results significantly. The map created is robust to illumination changes and it was interesting to check that the algorithm works during night-time.

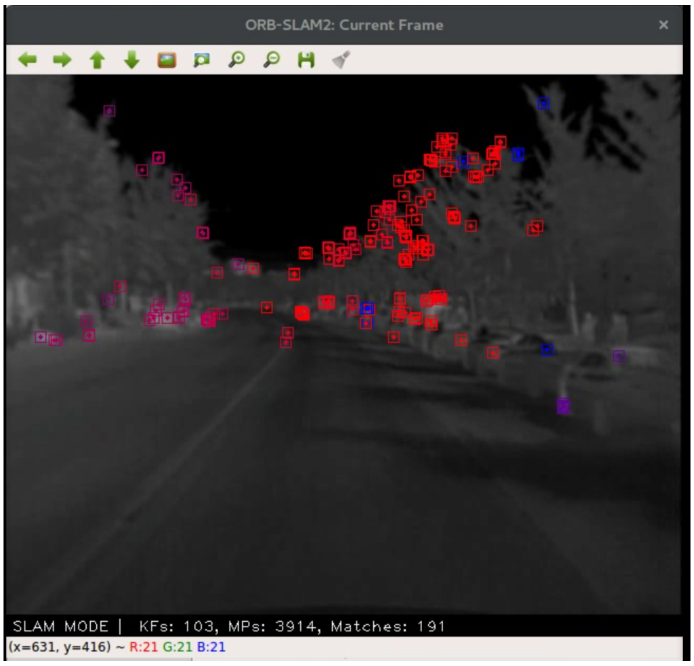

ORB-SLAM2: Current Frame
SLAM MODE | KFs: 103, MPs: 3914, Matches: 191
(x=631, y=416) ~ R:21 G:21 B:21

FIGURE 4.2 – Current frame.

Figure 4.2 shows the current frame given by the camera with the corners showed with a figure. Colouring the current frame made the system to be slower. Thus, in the final application this feature was turned off.

4.2 MAP

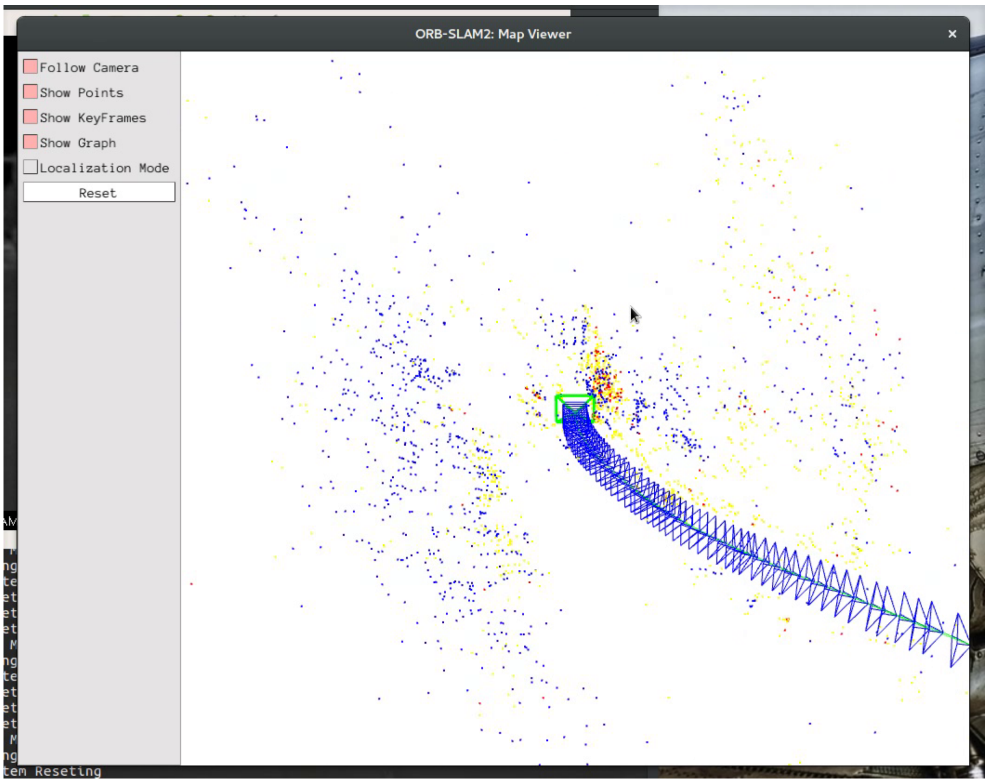
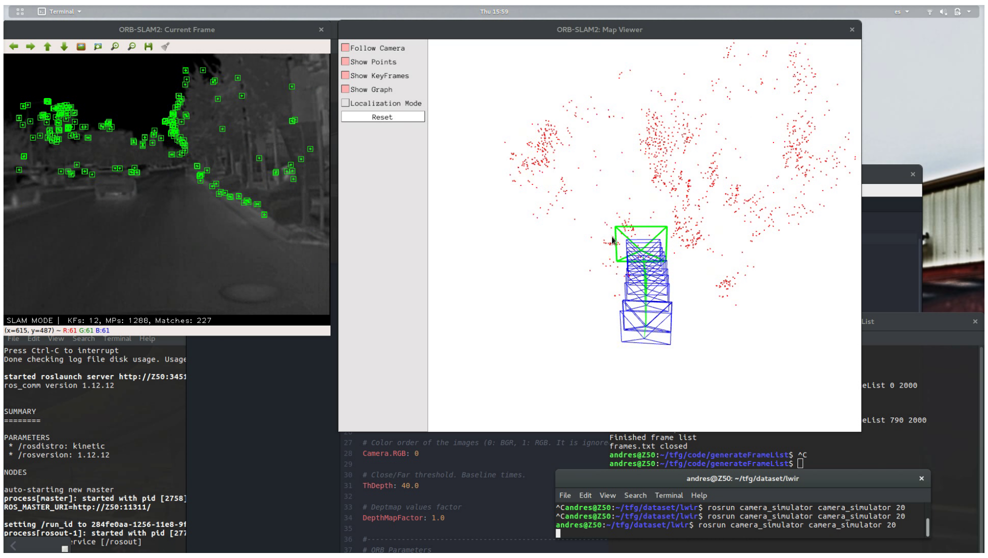


Figure 4.3 – Point cloud Map.

The viewer of the map works correctly according the algorithm described. The Point-Cloud is sparse, which means that there are not so many points, this is normal for a monocular camera. Although valuable information can be extracted from the map.



```
SLAM MODE |  KFs: 12, MPs: 1288, Matches: 227
(x=615, y=487) ~ R:61 G:61 B:61
```

```
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage

started roslaunch server http://Z50:3451
ros_comm version 1.12.12


SUMMARY
========

PARAMETERS
 * /rosdistro: kinetic
 * /rosversion: 1.12.12

NODES

auto-starting new master
process[master]: started with pid [2758]
ROS_MASTER_URI=http://Z50:11311/

setting /run_id to 284fe0aa-1256-11e8-9f
process[rosout-1]: started with pid [277
                  rvice [/rosout]
```

```
# Color order of the images (0: BGR, 1: RGB. It is ignore
Camera.RGB: 0

# Close/Far threshold. Baseline times.
ThDepth: 40.0

# Deptmap values factor
DepthMapFactor: 1.0

#------------------------------------------------
# ORB Parameters
```

```
eList 0 2000

eList 790 2000
Finished frame list
frames.txt closed
andres@Z50:~/tfg/code/generateFrameList$ ^C
andres@Z50:~/tfg/code/generateFrameList$
```

```
andres@Z50: ~/tfg/dataset/lwir
^Candres@Z50:~/tfg/dataset/lwir$ rosrun camera_simulator camera_simulator 20
^Candres@Z50:~/tfg/dataset/lwir$ rosrun camera_simulator camera_simulator 20
andres@Z50:~/tfg/dataset/lwir$ rosrun camera_simulator camera_simulator 20
```

Figure 4.4 – Full Mono node.

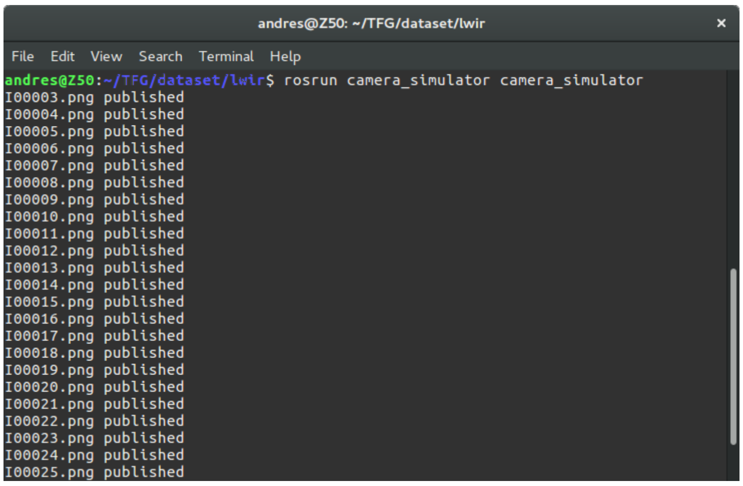
4.3 PUBLISHING FRAMES

Besides, I wrote a ROS package that simulates a camera publishing frames on a topic. *camera_simulator* takes a list of images from *frames.txt* and if you wish you can specify the fps. This file was created beforehand with *generateFrameList* that I coded to have the format from KAIST dataset given the starting and ending frame if you wish.

```
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <opencv2/highgui/highgui.hpp>
#include <cv_bridge/cv_bridge.h>
#include <string>
#include <fstream>
#include <iostream>
#include <stdlib.h>
int main(int argc, char** argv)
{
 int fps = 20;
 if(argc>1) fps =atoi(argv[1]);
 ros::init(argc, argv, "image_publisher");
 ros::NodeHandle nh;
 image_transport::ImageTransport it(nh);
 image_transport::Publisher pub =
 it.advertise("FLIR/image", 1);
 std::ifstream file("frames.txt");

 if(file.is_open()){
 std::string namefile;
 ros::Rate loop_rate(fps);
 while (nh.ok()) {
  if(!file.eof()){
  file >> namefile;

  cv::Mat image = cv::imread(namefile, CV_LOAD_IMAGE_COLOR);
  sensor_msgs::ImagePtr msg = cv_bridge::CvImage(std_msgs::
  Header(), "bgr8", image).toImageMsg();
  std::cout << namefile << " published" <<std::endl;

  pub.publish(msg);
  ros::spinOnce();
 }
   loop_rate.sleep();
 }
 file.close();
}
}
```

In addition, I created a *package.xml* and a *CMakeLists.txt*. Everything was placed correctly in my catkin workspace and then I ran the *catkin_make* to compile and install the code. Given that roscore is running, we can start publishing frames by running the following command in our image folder :

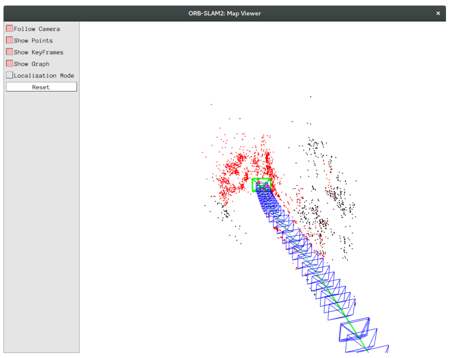
```
andres@Z50:~$ rosrun camera_simulator camera_simulator 20
```

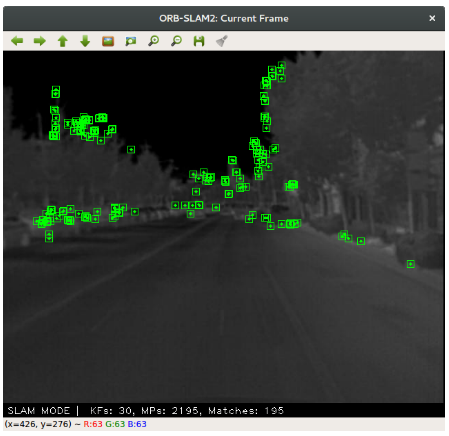

andres@Z50: ~/TFG/dataset/lwir
File Edit View Search Terminal Help

```
andres@Z50:~/TFG/dataset/lwir$ rosrun camera_simulator camera_simulator
I00003.png published
I00004.png published
I00005.png published
I00006.png published
I00007.png published
I00008.png published
I00009.png published
I00010.png published
I00011.png published
I00012.png published
I00013.png published
I00014.png published
I00015.png published
I00016.png published
I00017.png published
I00018.png published
I00019.png published
I00020.png published
I00021.png published
I00022.png published
I00023.png published
I00024.png published
I00025.png published
```

FIGURE 4.5 – Running camera_simulator.

Once the images are being published ORB-SLAM 2 will receive them as it is subscribed to the same topic.





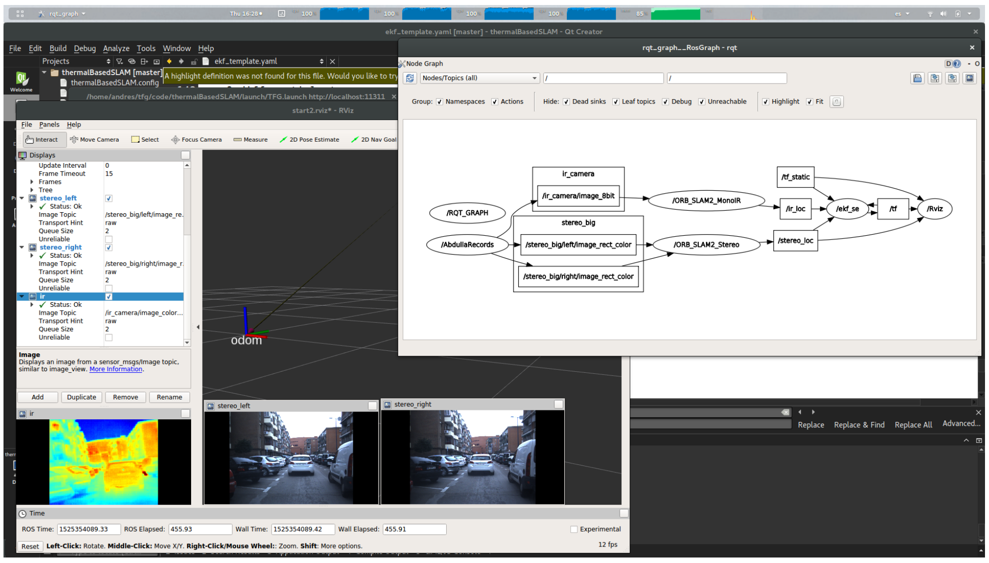
```
SLAM MODE |  KFs: 30, MPs: 2195, Matches: 195
(x=426, y=276) ~ R:63 G:63 B:63
```

4.4 UC3M MATERIAL

4.4.1 *Thermal Camera node*

Initialization was possible with the Thermal camera but pure rotations made the SLAM to get lost quite often. No camera calibration was available so the map was not accurate. ORB SLAM 2 documentation states that if a low resolution camera is used the conditions for initialization can be relaxed from the calibration files but the results show that the initialization is weaker. Initialization threshold was lowered to 10 to make initialization possible.

```
# ORB Extractor: Fast threshold
#Image is divided in a grid. At each cell FAST are extracted
imposing a minimum response.
# Firstly we impose iniThFAST. If no corners are detected
we impose a lower value minThFAST
#You can lower these values if your images have low contrast
ORBextractor.iniThFAST: 10
ORBextractor.minThFAST: 7
```



The stereo localization node generates a map sooner than the monocular node. It has a denser point cloud than the monocular node. Robot localization starts correctly once the system is initialized.

CHAPTER 5

Conclusions and future works

The estimate for the position is promising but the results are not the best that could be obtained, the reason is that a calibration matrix for the thermal camera would be needed plus a transformation frame to relate both cameras. Another aspect to be taken into account is the initialization from both cameras, ORB SLAM 2 initializes the frames in an arbitrary frame which means that both nodes are not related, not even in scale of maps. A global map processing would be helpful to give solution to nodes getting lost.

A common node with the three cameras could have been implemented to gain efficiency. Although, the expertise on this field and material for testing were not sufficient to tackle the problem in that way, an initial implementation has been done.

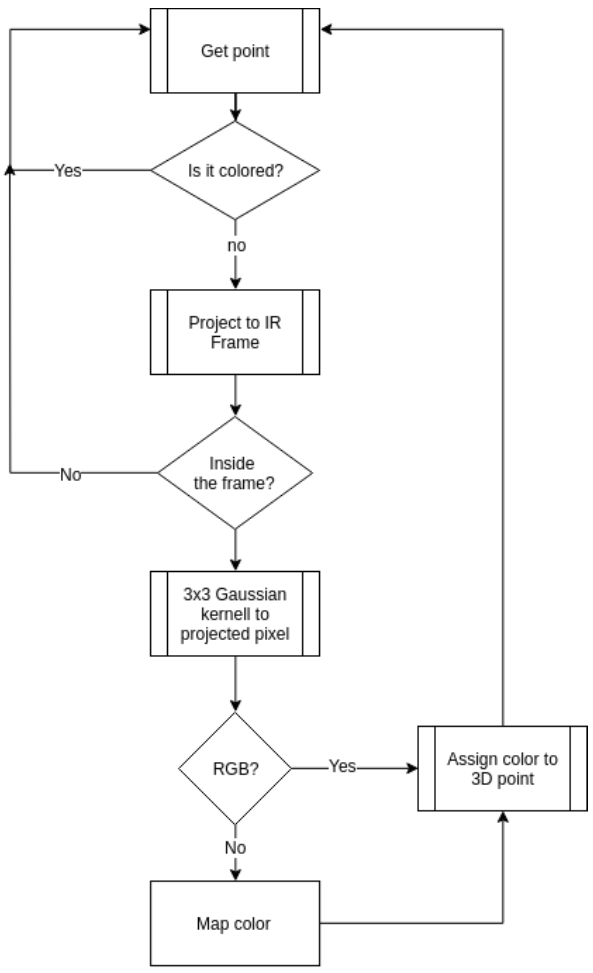
5.1 Future works

ORB SLAM 2 provides a Vocabulary file as explained before. That vocabulary was obtained from images taken from a RGB camera in urban setting which do not necessarily have to match the vocabulary from the thermal camera. Moreover, if SLAM is performed by an aerial vehicle the trained environment differs from the application environment probably reducing the matches. This field should be tested.

In this project, SLAM did not have a probabilistic model to work with making the covariances unknown. Tuning these covariances could make improved the estimate.

An algorithm was considered when exploring the possibilities of the Thermal camera. This package would bring the possibility to colour a pointcloud given the pose of the camera in the coordinate frame of such pointcloud. The node projects the 3D point by means of the camera calibration matrix and if the point lands on the frame the value of the pixel will give the temperature of the point.

This implementation could be applied also with RGB cameras for LIDAR PC colouring



5.1.1 *Input from MAVROS (GPS, IMU, input vector)*

The system is ready to add more inputs to improve the estimate. This local and continuous data is valuable when the nodes loose tracking and resets. This problem harms loop closing which is, as a matter of fact, one of the key features in a SLAM algorithm.

I

Appendices

APPENDIX A

Software needed

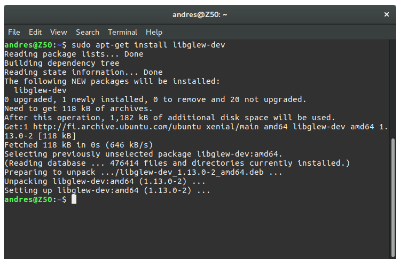
For this project ROS, OpenCV, robot localization and ORB-SLAM were needed.

A.1 ORB SLAM 2

A.1.1 *Pangolin*

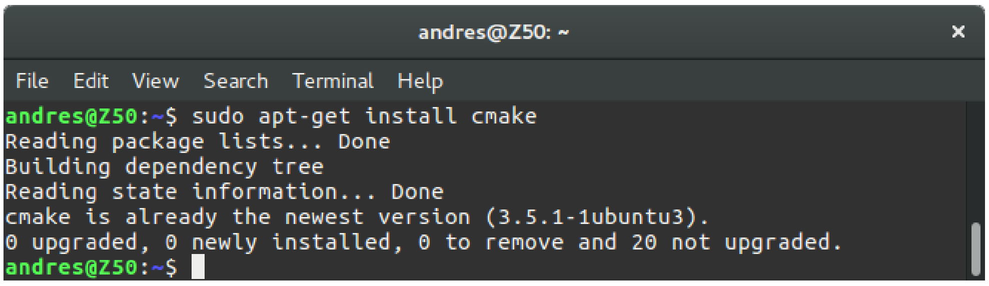
Pangolin is a lightweight library that manages OpenGL in the visualization part of ORB-SLAM or ORB-SLAM2.

We first install Glew.



```
andres@Z50:~$ sudo apt-get install libglew-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  libglew-dev
0 upgraded, 1 newly installed, 0 to remove and 20 not upgraded.
Need to get 118 kB of archives.
After this operation, 1,182 kB of additional disk space will be used.
Get:1 http://fi.archive.ubuntu.com/ubuntu xenial/main amd64 libglew-dev amd64 1.
13.0-2 [118 kB]
Fetched 118 kB in 0s (646 kB/s)
Selecting previously unselected package libglew-dev:amd64.
(Reading database ... 476414 files and directories currently installed.)
Preparing to unpack .../libglew-dev_1.13.0-2_amd64.deb ...
Unpacking libglew-dev:amd64 (1.13.0-2) ...
Setting up libglew-dev:amd64 (1.13.0-2) ...
andres@Z50:~$
```

CMake is also needed. But it was already installed.



```
andres@Z50:~$ sudo apt-get install cmake
Reading package lists... Done
Building dependency tree
Reading state information... Done
cmake is already the newest version (3.5.1-1ubuntu3).
0 upgraded, 0 newly installed, 0 to remove and 20 not upgraded.
andres@Z50:~$
```

And finally Pangolin can be installed from source:

```
git clone https://github.com/stevenlovegrove/Pangolin.git
cd Pangolin
mkdir build
cd build
cmake ..
cmake --build .
```

A.1.2 *Eigen3*

Eigen3 is a library that allows you to operate with matrix including SVD. This library is needed by G20. To install it I downloaded it from the official website, uncompressed it and installed it as follows

```
tar -zxvf eigen-eigen-5a0156e40feb.tar.gz
mkdir build
cd build
cmake ../eigen-eigen-b9cd8366d4e8
sudo make install
```

A.1.3 *OpenCV*

OpenCV is an open source library for computer vision, it can be ran in C++ as well as Python.

```
sudo apt-get install cmake git libgtk2.0-dev
pkg-config libavcodec-dev libavformat-dev libswscale-dev

git clone https://github.com/opencv/opencv.git

unzip opencv

cd opencv

mkdir build

cd build

cmake -D CMAKE_BUILD_TYPE=Release -D
CMAKE_INSTALL_PREFIX=/usr/local ..

make -j7

sudo make install
```

A.1.4 *ORB-SLAM 2 installation*

```
git clone https://github.com/raulmur/ORB_SLAM2.git ORB_SLAM2

cd ORB_SLAM2
```

```
chmod +x build.sh

./build.sh

chmod +x build_ros.sh

./build_ros.sh
```

We also have to source the files, this can be done automatically if we put it in .bashrc

A.2 ROS

```
sudo sh -c 'echo "deb http://packages.ros.org/
ros/ubuntu $(lsb_release -sc) main" > /etc/
apt/sources.list.d/ros-latest.list'

sudo apt-key adv --keyserver hkp://ha.pool
.sks-keyservers.net:80 --recv-key 421C365B
D9FF1F717815A3895523BAEEB01FA116

sudo apt-get update

sudo apt-get install ros-kinetic-desktop-full
```

Now we have to configure ros.

```
sudo rosdep init

rosdep update

echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc

source ~/.bashrc
```

Bibliography

[1] Edited figure. 2013. URL: https://tex.stackexchange.com/questions/96074/more-elegant-way-to-achieve-this-same-camera-perspective-projection-model.

[2] Yaakov Bar-Shalom, Thiagalingam Kirubarajan, and Xiao-Rong Li. *Estimation with applications to tracking and navigation:* Wiley, 2007.

[3] H. Budzier and G. Gerlach. "Calibration of uncooled thermal infrared cameras". In: *Journal of Sensors and Sensor Systems* 4.1 (Feb. 2015), pp. 187–197. DOI: 10.5194/jsss-4-187-2015.

[4] Fintan Corrigan. *10 Thermal Vision Cameras For Drones And How Thermal Imaging Works*. May 2018. URL: https://www.dronezon.com/learn-about-drones-quadcopters/9-heat-vision-cameras-for-drones-and-how-thermal-imaging-works/.

[5] Rikke Gade and Thomas B. Moeslund. "Thermal cameras and applications: a survey". In: *Machine Vision and Applications* 25.1 (Sept. 2013), pp. 245–262. DOI: 10.1007/s00138-013-0570-5.

[6] A. Garcia-Cerezo et al. "Development of ALACRANE: A Mobile Robotic Assistance for Exploration and Rescue Missions". In: *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*. Sept. 2007, pp. 1–6. DOI: 10.1109/SSRR.2007.4381269.

[7] K. Hajebi and J. S. Zelek. "Structure from Infrared Stereo Images". In: *2008 Canadian Conference on Computer and Robot Vision*. May 2008, pp. 105–112. DOI: 10.1109/CRV.2008.9.

[8] Chris Harris and Mike Stephens. "A combined corner and edge detector". In: (Jan. 1988), p. 50.

[9] *JARUS*. URL: http://jarus-rpas.org/.

[10] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262. DOI: 10.1109/TRO.2017.2705103.

[11] Takashi Ogura. *cv_camera*. URL: http://wiki.ros.org/cv_camera.

[12] E. Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International Conference on Computer Vision*. Nov. 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[13] Richard Szeliski. *Computer vision: algorithms and applications*. Springer, 2011.

[14] Juan D. Tardós. “ORB-SLAM: a Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.

[15] S. El-Tawab, M. Abuelela, and Yan Gongjun. “Real-time weather notification system using intelligent vehicles and smart sensors”. In: *2009 IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*. Oct. 2009, pp. 627–632. DOI: 10.1109/MOBHOC.2009.5336947.

[16] Yohann Tendero, Stéphane Landeau, and Jérôme Gilles. “Non-uniformity Correction of Infrared Images by Midway Equalization”. In: *Image Processing On Line* 2 (Dec. 2012), pp. 134–146. DOI: 10.5201/ipol.2012.glmt-mire.

[17] S. Vidas, P. Moghadam, and S. Sridharan. “Real-Time Mobile 3D Temperature Mapping”. In: *IEEE Sensors Journal* 15.2 (Feb. 2015), pp. 1145–1152. ISSN: 1530-437X. DOI: 10.1109/JSEN.2014.2360709.

[18] S. Vidas and S. Sridharan. “Hand-held monocular SLAM in thermal-infrared”. In: *2012 12th International Conference on Control Automation Robotics Vision (ICARCV)*. Dec. 2012, pp. 859–864. DOI: 10.1109/ICARCV.2012.6485270.

[19] Arto Visala. *STATE ESTIMATION FOR NONLINEAR DYNAMIC SYSTEMS*.