SoViN: a Software platform for Visual Navigation

Laurent Lequièvre, Jonathan Courbon, Youcef Mezouar and Philippe Martinet

LASMEA
24, avenue des Landais
63177 AUBIERE

Abstract

In this paper, we present a software platform (SoViN) dedicated to visual memory management and vision-based navigation of autonomous vehicles. This software allows to achieve navigation tasks in large scale environments using natural landmarks. It has especially been designed to prototype visual memory-based strategies. Such approaches have the major advantage that only key views and related image descriptors are stored. This process is thus expected to be efficient by means of 1) memory needed to store data and 2) computational cost. These points are crucial issues for real-time navigation in large scale environment. We will see that SoViN allows to meet these expectations.

Keywords

Mobile Robot; Visual memory-based navigation; Software architecture; Database access and management; HMI; large scale environment; Real-time application

1 Introduction

Automatic navigation can be seen as a four steps process: 1) map building, 2) localization onto the map, 3) path planning and 4) control to actually achieve the navigation task. Many works deal with the problems of fuzzing steps 1) and 2) on a single stage (Simultaneous Localization And Mapping; SLAM). Briefly, such an approach consists generally on comparing current sensors data to the predicted ones and then, to update both the map and the position of the robot. In that aim, most of the strategies are based on visual sensors or on range sensors. Unfortunately, even if computers are more and more powerfull, those strategies are restricted to small environments since the computational cost highly increases with the number of features integrated onto the map.
In this paper, we particularly focus on emerging navigation strategies using visual sensors only. The main idea is to represent the mobile robot environment with a bounded quantity of images gathered in a database (visual memory). For example, [10] proposes to use a sequence of images recorded during a human teleoperated motion, and called View-Sequenced Route Reference. Such a strategy is called "mapless" (refer to [4]). Indeed, any notion of map nor topology of the environment appears, neither to build the reference set of images, nor for the automatic guidance of the mobile robot. Similar approaches have been proposed for urban vehicles in [17, 5]. The visual memory can also be topologically organized if images or sequence of images are linked to a notion of places

as in [7]. Finally, such an approach may be enhanced by metric information such as the position of 3D points reconstructed from the images (refer for example to [12]). Current applications of these approaches are restricted to small scale environments (navigation task along trajectories no longer than 500 meters) essentially, because of inefficient memory management. In this paper, we describe a Software for Visual navigation: SoViN. This software allows easy memory management (upload, update, removal), memory visualisation and real-time navigation. This software is sufficiently generic to allow the prototyping of different visual memory-based navigation methods. In Section 2, four typical visual navigation strategies which can be implemented using SoViN are described and Section 3 presents the Software architecture. Experimentation's carried on with an urban vehicle are finally presented in Section 4.

2 Visual navigation strategies

The considered vision-based navigation strategies rely on two steps. The first step consists on building off-line the visual memory. The robot is first teleoperated along paths and video sequences are acquired. Key images are extracted from the video sequences and stored (visual memory). The second step is realized on-line. First, the robot localizes itself in the visual memory. A visual path is then extracted from the visual memory in order to reach a desired key image from an initial one. Finally, control outputs are computed to follow this visual path. As examples, the different steps of four approaches which can be implemented using SoViN are described in the sequel: in [7], a wheel chair navigates in an indoor environment with an embedded omnidirectional camera. In [12] and in [5], an urban vehicle follows a visual path. While a 3D reconstruction of the full path is computed in [12], only a local reconstruction is used in [5]. In [1, 2], a complete vision-based navigation framework is proposed for indoor or outdoor environment, with a perspective or with an omnidirectional camera.

2.1 Off-line memory building

In [12, 5, 2], a perspective camera is embedded onto the urban vehicle and looks forward while in [7] an omnidirectional camera pointing to the ceil is employed.

Key images selection In order to reduce the complexity of the images sequences, only key views are stored and indexed on a visual path. This step is realized manually or automatically. In almost all the recent approaches, points are used as visual features. They are detected and described by a descriptor which is used to match points of two images. The control law is computed from matched points between the current image $\mathcal{I}_c$ acquired by the embedded camera and the desired key image to reach $\mathcal{I}_{n+1}$. It is thus necessary to track those points from the key image $\mathcal{I}_{n-1}$ to the following key image $\mathcal{I}_n$. In [12, 2], points are detected with the Harris corner detector [8]. The matching scores between points of those two images are computed with a Zero Normalized Cross Correlation (ZNCC). This method is illumination invariant and its computational cost is small. A new key image $\mathcal{I}_n$ is selected if: a) there are as many images as possible between $\mathcal{I}_n$ and $\mathcal{I}_{n-1}$, b) there are at least N point correspondences between $\mathcal{I}_{n-1}$ and $\mathcal{I}_n$ and c) there are at least M point correspondences between $\mathcal{I}_{n-2}$ and $\mathcal{I}_n$. This criterion ensures that there are common matches at least in three consecutive views. Finally, a partial 3D reconstruction using the epipolar geometry is computed with the calibrated 5-point algorithm [11] coupled to RANSAC algorithm [6]. This last step allows to keep only robust matchings. In [5], Harris corners are extracted in the key image $\mathcal{I}_{n-1}$ and initialize a modified Kanade-Lucas-Tomasi (KLT) feature tracker [13]. A partial 3D reconstruction is realized with the 5-point algorithm [11] coupled to MLESAC random sampling algorithm [16]. A new key image is selected when the number of matched points is low or when the reconstruction error is high. In [7], rotation reduced and color enhanced SIFT features [9] are extracted in the first image. Those points are tracked with the KLT tracker.

Visual memory organisation The next step consists on organizing the key images. The visual memory can be organized as a single oriented edge (by analogy to the graph theory) containing the successive key images or can be composed of multiple edges. In this last case, the visual memory is organized as a graph, where each edge is defined as a set of ordered images as proposed in [7, 2]. Some suplementaries informations are generally added to the stored images: 2D points robustly matched between two key images in [5, 2], robust 2D points, related 3D coordinates and camera poses in [12], 3D position of the visual features in [7].

2.2 On-line navigation

The on-line navigation can be divided on three main steps: initial localization, path-planning and path following. The localization consists on finding the key image of the memory which best fits the current image acquired by the embedded camera. A visual path (*i.e.* a succession of key images) is extracted from the memory in order to link the initial key image to a desired key image. Finally, the robot is controlled in *real-time* along the visual path.

Initial localization The initial localization is realized manually or automatically. In [5], the user selects a reference image close to the robot's current location. For automatic techniques, the initial localization is obtained by comparing each key image to the current image. This step requires few seconds but it is only performed once. In [12], this is achieved by matching interest points between the two images and computing a camera pose with RANSAC. The pose obtained with the higher number of inliers is considered as a good estimation of the camera pose for the first image. In [3], a hierarchical localization process is proposed. In a first step, only some key images are selected by matching the global descriptor of the current image to the global descriptors of every key image. The computational cost of this step is low. To obtain the initial localization, an approach similar to [12] is then used, but only for the selected key images. This second step increases the accuracy and the robustness of the localization process.

Path-building The path building step consists on defining a path allowing the robot to reach a desired configuration from a current one. It can also consists on finding a set of images linking the current to the desired images since in the considered approaches the robot configurations are associated to images. When the visual memory is composed of a single edge, it is then straightforward to solve this problem. When many edges compose the visual memory, this step can be done manually by selecting the edges to follow or automatically by using efficient search algorithm such as Dijkstra's algorithm.

Path following The path following step can be splitted in two main stages: the estimation of the state of the robot (sometimes called *localization*) and the computation of the control law. The state is estimated from the current image, the desired image of the memory and eventually the former key image when 3 views are employed for 3D reconstruction purposes. In [12], the position and orientation of the current camera frame is computed in an absolute frame related to the visual path. The state of the robot is the position of the robot frame with respect to the trajectory the robot has travelled along during the off-line step. Finally, a control law, adapted to the non-holonomy of the vehicle is computed from this position. This control law is based on the chained system theory [15]. In [3], the state is chosen as the scaled displacement between the current camera frame and the desired camera key frame. A similar control law as the one proposed [15] is used. In [5], the state is defined as the error between the centroids of a set of points matched in the current image and the desired key image. The rotational velocity is then defined as proportional to the error on centroids. In [7], a homing strategy is used.

2.3 Requirements

To achieve visual memory-based navigation tasks several more or less basic tools are necessary. First, navigation tasks makes intensive use of image and data processing such as interest points extraction, matching and tracking. It also employs automatic image selection from video sequences (importation step) generally based on matching scores. 3D reconstruction algorithm can also be usefull for robust matching as well as 3D localisation when needed. Those tools are used both in on-line and off-line stages to select key images and estimate 2D points with their position and their descriptors, 3D points, camera positions, etc . . . .

To be efficient, one have also to make use of specific tools for organisation and management of those data. When small environments are considered data can be directly downloaded in RAM while it is not the case when offline and online steps occur at different time or when the amount of data is too important. Data have to be stored into a well structured data-base allowing fast reading. Note that data uploading may occur in real-time during the autonomous navigation. As explained before, the map is generally a graph of edges where each edge contains successive images acquired by a camera. 2D points belong to an image and may correspond to 3D points if they have been matched. The design of the database must agree this structure and its access must be sufficiently fast to allow real-time processing (near the video rate). Finally, for debug purposes as well as to check and interact in real-time with the navigation process an HMI is necessary. In the sequel, the software for Visual Navigation (SoViN) developed to fulfill those requirements is detailled.

3 Overview of SoViN architecture

The architecture of SoViN (Software for Visual Navigation) is summarized in Figure 1. Basically, it can be decomposed on three parts: a library for database access and management, a library for processing and an Human Machine Interface. Those three parts are detailled in the sequel.

3.1 Data models

The database is structured using the full-integrated conception technique MERISE. First the Conceptual Data Model (CDM) has been designed in order to define the required entities and their

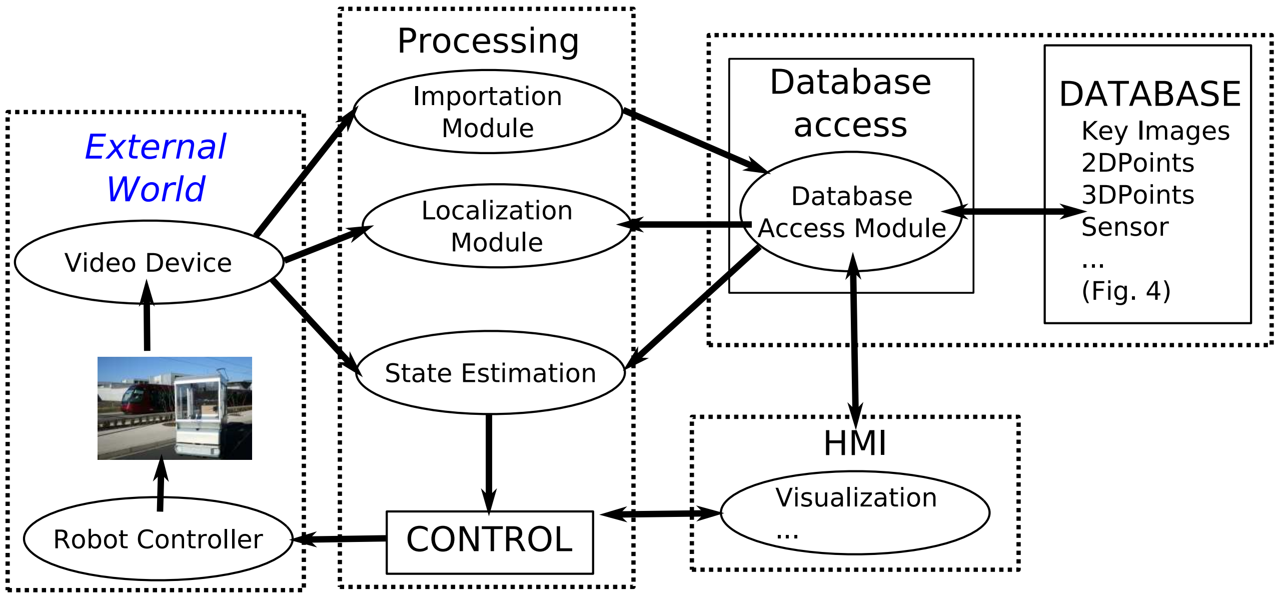


FIG. 1: SoViN is composed of three main parts: processing (libSovinProcessing), data storage access library (libSovinBDD) and Human Machine Interface (SovinHMI).

relations. The physical model of SoViN is then obtained from the CDM and used to design the physical Sovin database. Finally, the object model, integrated to libSovinBDD library, has been developed to manage this database.

A- Sovin Conceptual Data Model

The CDM represents a conceptual description of the data. This model was designed for SoViN to fulfill the requirements of most of the vision-based navigation frameworks. In this model, entities represent structured and organized data for storage in data management systems. Each entity is identified by a specific attribute called key or 'ID'. The relations link different entities (n-ary relations or *cardinality*). This model with the entities, their major attributes and the relations between entities is presented in Fig. 2 and detailled in the sequel.

A TEST represents a set composed of several EDGEs (cardinality 0,n) or paths. An EDGE represents a path acquired during a teleoperated step. It can have none or n following EDGEs (relation "Has for following Edge"). An EDGE contains one or several NODEs (cardinality 1,n). A NODE represents a position of the robot where an image was acquired. The position of a NODE in an absolute reference frame is not necessary. However, it exists a relation between two successive positions: a NODE has (or dot not have) a following NODE (relation "Has for following Node"). Note that an EDGE must be linked at least to one NODE since it starts and finishes in a NODE (cardinality 1,1). As an example, let us consider an urban environment. A TEST can represent a district, an EDGE a street or a part of a street. From a street, it is possible to move to different other streets. In a street, images are acquired at successive NODEs.
Let consider that multiple cameras have been used for a TEST. At a NODE, an image or multiple images are acquired by the sensors. Several images may be acquired at this NODE but those images must be acquired by different sensors. This condition is fullfilled by the relation "Contains

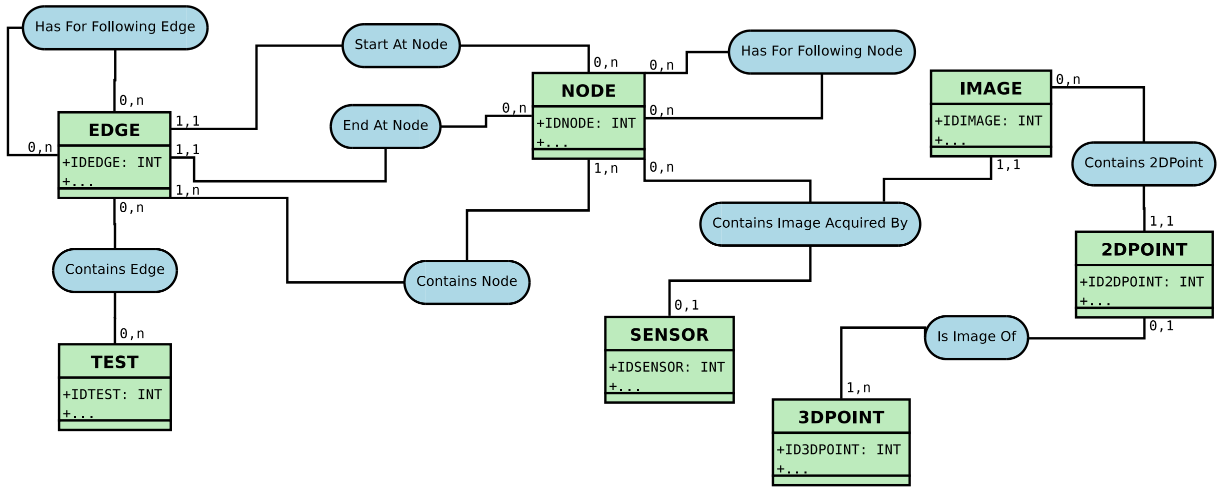


FIG. 2: Conceptual Data Model of Sovin Database

Image Acquired By". The entity "IMAGE" has several attributes like positions and orientations coordinates and contains the image (*i.e.* the color or black and white picture acquired by the camera).
An other important element is the image points (2DPOINTS) and the 3D points (3DPOINTS). Those entities are part of the CDM. Each image can contain 2D points (cardinality 0,n), and each 2D point may be or not the image of a single 3D point (cardinality 0,1).
In this model, the n-ary relations (with their cardinalities) are readable in both directions. For example, a 3D point exists if and only if a 2D point is the image of it, but this 3D point may be the origin of several 2D points (cardinality 1,n).

B- Sovin Physical Data Model

The physical data model (PDM) defines the implementation of the physical structures of the database (refer to Fig. 3). The PDM has been obtained from the conceptual model.
Each entity of the conceptual model is expressed into a table [1] of the physical model and each attribute is converted into column of the table. An element of an entity is stored as a row of the corresponding table. As every row must be unique, an unique identifying integer named key (primary key) is created for each row of the table. The primary keys are integers with the "AUTO_INCREMENT" property, allowing that the data management system provides us a unique key which increases progressively. Of course, primary keys cannot be null. For instance, nodes are saved in the table **NODE** and each node contains a key `IDNODE`. The expression of the n-ary relations depends on the cardinalities. For simple cardinalities, the relationships between tables is converted into foreign keys (which are keys from other tables). Some n-ary relations impose to create intermediate physical tables. This is the case when it is possible to have multiple elements of a table linked to the element of an other table (cardinality (*,n)). For example, a test contains none or n edges. This condition imposes to create an intermediate entity (table) **Contains_Edge** in the physical model of Sovin Database. The elements of the new table have to contain the key of the element of **Test** (`IDTEST`) and the key of the element of **Edge** (`IDEDGE`).

[1] In the sequel of this article, a table is written in bold font.

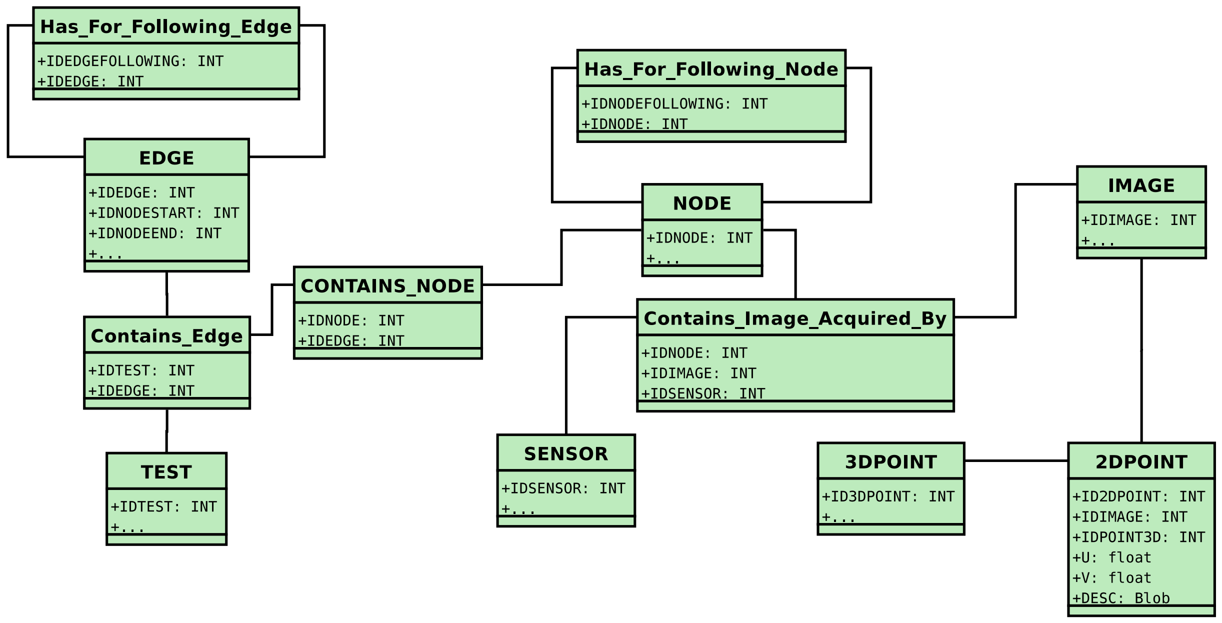


FIG. 3: Physical Data Model of Sovin Database

Each table is composed of keys and attributes. Relational database management systems support a number of data types: numeric types (integer, float, double, boolean ...), date and time types (date, datetime, time ...), and string types (varchar, text, blob[2] ...).
For instance, the **2DPOINT** table has for attributes (or columns) the key `IDPOINT2D` (integer) but also the elements of a 2D point: the coordinates `U` and `V` (float), the neighbourhood descriptor `DESC` (blob) and the key of the image `IDIMAGE` the point belongs to. It should be noticed that table **2DPOINT** contains the key `IDIMAGE` of the table **IMAGE** because of the (1,1) cardinality of the **Contains 2D Point** relation.

C- Sovin Database

Once the PDM has been designed, it is necessary to create the database. Later, data will be added, manipulated and selected. In that aim, SQL (Structured query language) is used. The SQL language can be use to specify data definition and to manage data manipulation. SQL scripts have been generated from the physical model to create the database. Once the database is generated, the SQL language is used to send requests to the database in order to insert, select, update or delete elements of the tables. The selection of data from one or several tables with some search criteria is realized thanks to the ’SELECT’ statement. This kind of complex requests would be very difficult to do with simple textual files while the SQL language managed it easily.
A database language standard like SQL is appropriate for all database management system. In Sovin, the database management system MySql has been chosen. This system natively supports different storage engines. This is an asset in our case because it is thus possible to choose the more adapted storage engine for each table in order to optimize its use. As our applications require fast readings (’SELECT’ queries), MyISAM has been chosen as storage engine for all our tables.

[2]Blob: binary large object

During a selection query, MySQL reads all the lines of the considered table successively, and each time, make the comparisons necessary to extract the relevant result. The larger the table is, the more expensive research cost is. To speed up information seeking, it is possible to add indexes on the keys of the tables. The indexes are used to find more quickly resulting rows from a table given a specific criteria. It is thus important to create the indexes linked to the selection criteria. When the research is based on the key, then it is necessary to build an index on this key.
The SQL script written in List. 1 is executed to create the table **2DPOINT**. This script has been generated from the physical model and indexes have been added. We recover the required parameters: the primary key `ID2DPOINT` cannot be null and is increased automatically by the data management system, the key of the image (`IDIMAGE`) cannot be null too, thanks to the cardinality (1,1) of the relation ”Contains 2D Point“ and finally the **3DPOINT** table key `ID3DPOINT` can be null thanks to the cardinality (0,1) of the relation ”Is Image Of“. We chose the type Blob (binary large object) for the descriptor of a 2D point, which allows us to store an array of M floats. Using the Blob type gives us more flexibility for the storage of the array, allowing us to save descriptors with different size. In our experiment a descriptor of size M=121 is used whereas other applications using for example SIFT descriptors will have to save descriptors with size M=128. A typical selection process with 2D points is the extraction of all the data of the points which belongs to a given image of key `IDIMAGE`=1. The SQL query is:
`SELECT ID2DPOINT, U, V, DESC FROM 2DPOINT WHERE IDIMAGE=1`. In order to decrease the selection time, an index *i_2dpoint_idimage* has been created onto the `IDIMAGE` attribute (refer to line 14 of List. 1). This index is used by MySQL for the former query.

Listing 1: SQL script for the creation of the table **2DPOINT**

```
1  create table 2DPOINT
2  (
3     ID2DPOINT            int not null auto_increment,
4     IDIMAGE              int not null,
5     ID3DPOINT            int,
6     U                    float not null,
7     V                    float not null,
8     DESC                 blob,
9     primary key (ID2DPOINT)
10 )
11 type = MyISAM;
12
13 create index i_2dpoint_idimage_id3dpoint on 2DPOINT(IDIMAGE,ID3DPOINT);
14 create index i_2dpoint_idimage on 2DPOINT(IDIMAGE);
15 create index i_2dpoint_id3dpoint on 2DPOINT(ID3DPOINT);
```

3.2 Database access and management library: LibSovinBDD

The Sovin software is written in C++. The Qt4 Library developed by Trolltech links the C++ code to MySql database. This library is free and has many functionalities to communicate with databases by using the Database Module. This module offers classes to access databases and send SQL queries to the database server. Drivers for all major databases like MySql are provided. Qt has also been chosen because it contains useful tools and classes to build graphical interfaces.
The functionalities of the libSovinBDD are classified in different directories (refer to Fig. 4). The *Databases* directory contains classes for low level requests on the tables of the database and access to the data. The *Management* directory contains classes for the high level management of the

database. The *Exceptions* directory contains the exceptions raised in case of errors while accessing the database. The properties of connection to the database are contained in a xml file and the directory *Xml* contains the classes for the reading of this file.
The directories *Databases* and *Management* are detailled in the following.

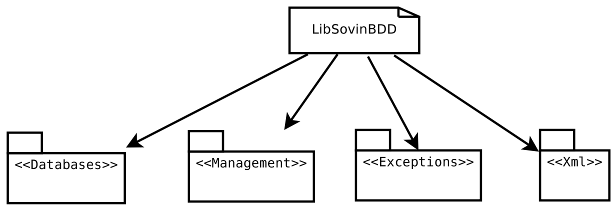


FIG. 4: Directories of LibSovinBDD

Databases : Sovin contains several classes to communicate with the database server. Each table is processed using two classes. The first class (called xTable where x is the name of the table) contains the requests which have to be sent to the table. The second class (called xRecord) represents a row of data (*i.e.* one element of the table). This class is used to read or modify data.
For instance, the class Point2DTable contains a method to retrieve all the 2D points (as a vector of Point2DRecord) of an image knowing its key:

```
static void getVectorPoint2D(const int idImage, QVector<Point2DRecord> & v);
```

It is then possible to modify or get data of each Point2DRecord thanks to the functions of this class as:

```
const float & getU() const throw(SovinInvalidDataException);
void setU(const float value);
const int & getIdImage() const throw(SovinInvalidDataException);
void setIdImage(const int id);
```

Note that the ’Table’ classes are interfaces allowing to send queries to the table. Consequently, this is not necessary to have several instances of these classes. By design, we have thus chosen to set all the methods of these classes static. Moreover, with this choice there is no dynamic allocation, which saves runtime.

Management : Managing data of Sovin is not a simple task. Each operation on data must preserve the integrity of the relational model. Several classes were added to carry out this process.
A first process is the data importation (class ’Importation’). It requires low level operations to be done along a correct order. For instance, when a new image has to be added to an edge, it is first necessary to create a node. This node is added to an edge and it is the following node of the previously imported node. Then, the image is added and is linked to the node and to a sensor.
A second process is the deletion of data (class ’Suppressions’). This is a complex task: it requires several low level operations described in ’Databases’ directory (functions `delete` in the classes xTable) as well as a more complex process in order to keep the structure valid. In that aim, specific functions have been designed in a class Suppressions for deleting each element while keeping a valid structure. As an example, the deletion of an image implies many changes in the structure (refer to List. 2). The image must be deleted from the table **Image**. It has to be deleted in the table **Contains Image Acquired By** too. Then, the 2D points associated to this image must

be deleted (cardinality (1,1) of relation Contains2DPoint). This suppression process needs functions of the Databases directory to retrieve elements of the table with the classes Point2DTable and Point3DTable, to get data of those elements with the class Point2DRecord. The classes xTable are also used to delete the element given by its id in the table (deletePoint3D, deletePoint2D). Of course, when removing a 2D point, the linked 3D point - if it exists - has to be removed if and only if this 3D point is not associated with another 2D point (refer to List. 3 for the deletion of a 2D point).

Some very useful functionalities needed by the experiments have been added. The ’Building’ class

Listing 2: Suppression of an image (and structure update)

```
void Suppressions::deleteImageInDepth(const int & idImage)
{
        // search 2D points key linked to the image
        QVector<int> vecIdPoint2D;
        ImageTable::getVectorIdPoint2D(idImage, vecIdPoint2D);

        // delete all the 2D points with 'deletePoint2DInDepth' method
        for(int i=0;i<vecIdPoint2D.size();i++)
                deletePoint2DInDepth(vecIdPoint2D[i]);

        // delete the Image contained in 'Contains Image Acquired By' table
        // use of low level method of class 'ImageTable' in 'Databases' directory
        ImageTable::deleteImageAcquiredBy(idImage);

        // delete the image
        // use of low level method of class 'ImageTable' in 'Databases' directory
        ImageTable::deleteImage(idImage);
}
```

Listing 3: Suppression of a 2D point (and structure update)

```
void Suppressions::deletePoint2DInDepth(const int & idPoint2d)
{
        // search 2D point by the key idPoint2d
        Point2DRecord point2d;
        Point2DTable::getPoint2DRecord(idPoint2d, point2d);

        // if 2D point is linked to a 3D point
        if(point2d.hasIdPoint3D())
        {
                // get the key of this 3D point
                int idPoint3D=point2d.getIdPoint3D();

                // if this 3D point is not associated with another 2D point
                if(Point3DTable::getNbPoint2D_HavingPoint3D(idPoint3D)==1)
                {
                        // delete the 3D point
                        Point3DTable::deletePoint3D(idPoint3D);
                }
        }

        // detete the 2D point
        Point2DTable::deletePoint2D(idPoint2d);

}
```

is dedicated to these operations. For example, a function removes nodes at the start or the end of

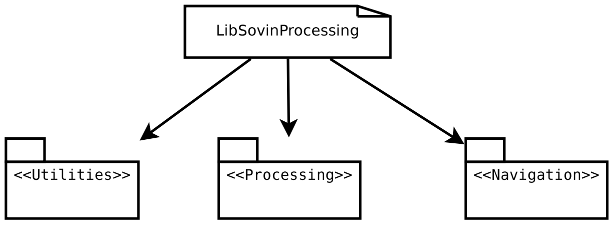


FIG. 5: Directories of LibSovinProcessing

an edge. Sometimes, it is required to cut an edge at a specific node to make a crossroad or to start a new edge at this node. In that aim, a function cuts an edge into two edges.

3.3 LibSovin Processing

Processing are composed of three main parts (refer to Fig. 5). The first one contains usefull classes: MathUtils (for processing functionalities) and DataConversion (for conversion of data between Qt+libSovinBDD objects and low level visual processing objects). The second one directory classes for the processing Importate and Localize. Finally, the third one contains the classes used during the on-line navigation step such as the visual path building and the visual path following processes.

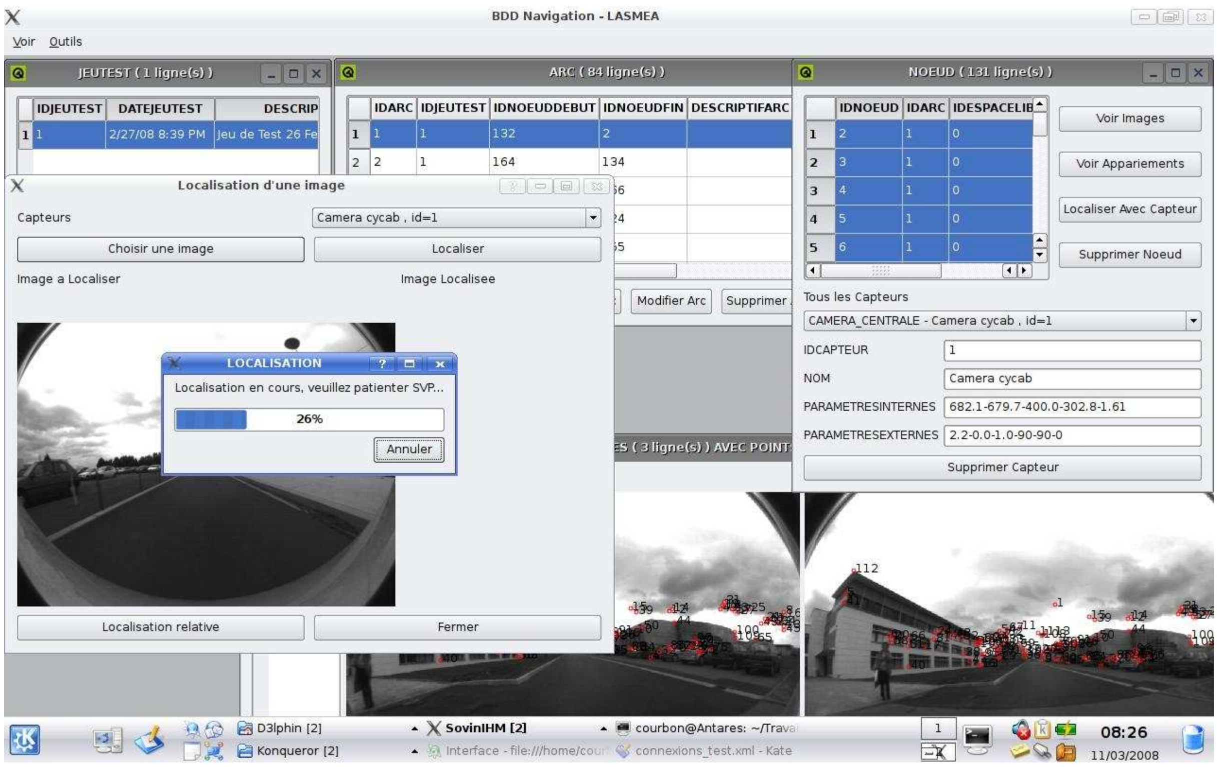
The computation of the number of 2D points matchings between an image of the database and a current image (refer to List. 4) is a typical example of processing function. It contains low-level functions to access the database, image processing function (from the visual processing library), data conversion functions and processing functions developed in LibSovinProcessing in order to manipulate the LibSovinBDD objects.

Listing 4: Matching of a current image to an image of the database

```
int getNbMatchingsBetweenImages(const int idMemImage, MCharImage CurrentImage)
HarrisDetector Detector;
vector<points2d> vecPoints2D;
QVector<Point2DRecord> vecCurrent2DPoints;
QVector<Point2DRecord> vecKey2DPoints;

// Detecte points in the current image
Detector.Detect(CurrentImage,vecPoints2D);
// Convert into LibSovinBDD object
DataConversion::convert2DPointsToVecPoint2DRec(vecPoints2D,vecCurrent2DPoints);

// load the points of the image of the memory
ImageTable::Load2DPointsFromIdImage(idMemImage, vecKey2DPoints);

// launch the points matching process
int nb_matching=Match(vecCurrent2DPoints,
                      vecKey2DPoints);
```

3.4 Human Machine Interface

A module for visualisation and high level actions control (HMI) has also been developed. It consists on a main window which is a MDI (Multiple Document Interface). The selection of the objects to visualize has been constrained in order to follow the structure of our model. It allows the graphical



	IDJEUTEST	DATEJEUTEST	DESCRIP
1	1	2/27/08 8:39 PM	Jeu de Test 26 Fe

	IDARC	IDJEUTEST	IDNOEUDDEBUT	IDNOEUDFIN	DESCRIPTIFARC
1	1	1	132	2	
2	2	1	164	134	

	IDNOEUD	IDARC	IDESPACELIB
1	2	1	0
2	3	1	0
3	4	1	0
4	5	1	0
5	6	1	0

Capteurs: Camera cycab , id=1

Localisation en cours, veuillez patienter SVP...

26%

Tous les Capteurs: CAMERA_CENTRALE - Camera cycab , id=1

IDCAPTEUR	1
NOM	Camera cycab
PARAMETRESINTERNES	682.1-679.7-400.0-302.8-1.61
PARAMETRESEXTERNES	2.2-0.0-1.0-90-90-0

FIG. 6: Overview of the Human Machine Interface of Sovin

management (update, delete ...) and representation of the database content (low level actions) as well as high level actions such as the localization of a given image in the images of the database. The design of the graphical part of this module has been facilitated by the use of the Qt library. The module uses the functionalities developed in the LibSovinBDD and LibSovinProcessing libraries. The main utility is the visualization of the information contained in the database (refer to Fig. 6). The HMI also allows the visualisation of a Test as a graph dynamically generated (using the GraphViz library). This graph represents the edges of a Test stored in Sovin database and a node of this graph represents an intersection between two edges (extracted from the table **Has for following edge**) (refer to Fig. 7). It also allows to easily check the feature detection and feature matchings algorithms results and to modify the database contents (for instance edge cutting (refer to Fig. 8)).

4 Experimentations

Our experimental vehicle is an urban electric car, named RobuCab, manufactured by Robosoft Company. It is depicted on Figure 9. Currently, RobuCab serves as development products in several French laboratories. The 4 DC motors are powered by lead-acid batteries, providing 2 hours autonomy. Vision and guidance algorithms are implemented in C^{++} language on a laptop using RTAI-Linux OS with a 2GHz Centrino Duo processor. The Fujinon fisheye lens, mounted onto a Marlin F131B camera, has a field-of-view of $185\deg$ and has been calibrated. The image resolution in the experiments was 800×600 pixels. The camera, looking forward, is situated at approximately 80cm from the ground. The parameters of the rigid transformation between the camera and the robot control frames are roughly estimated. Grey level images are acquired at a

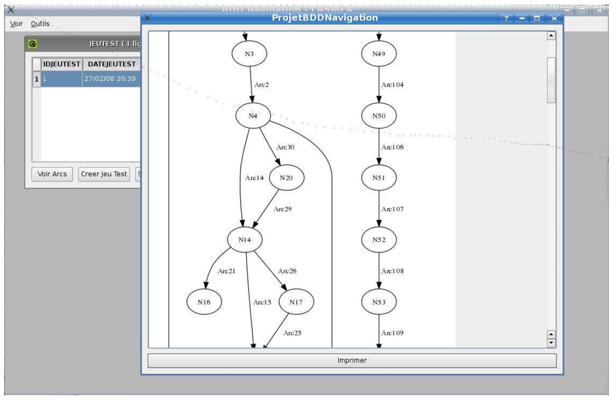
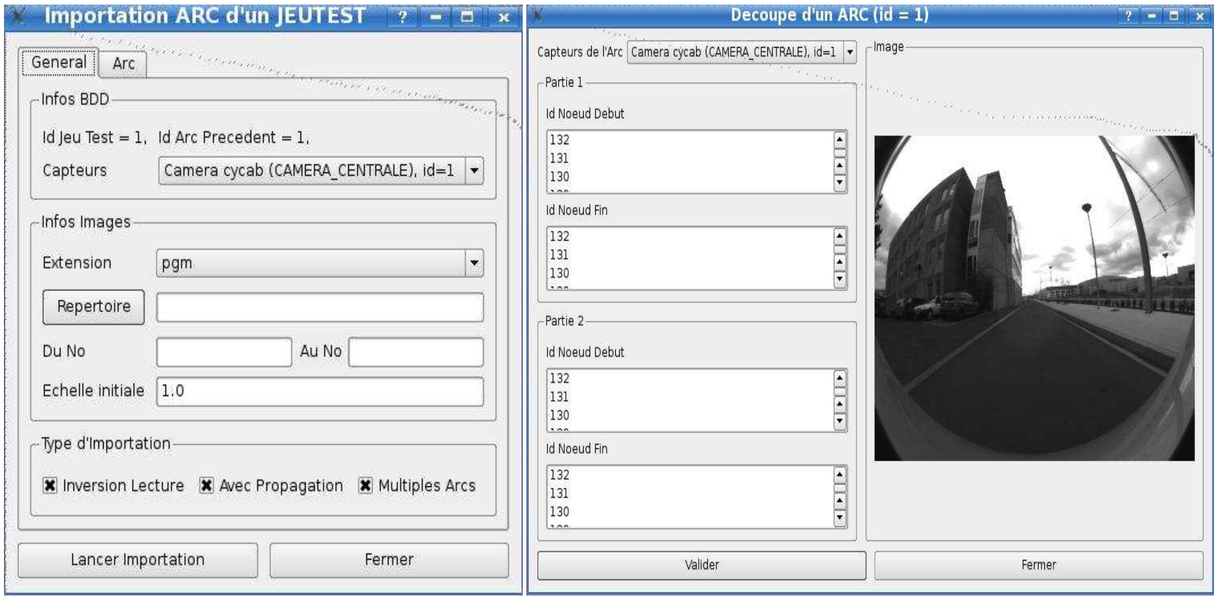


FIG. 7: Part of the graph representing the visual memory of a test of SoViN


Importation ARC d'un JEUTEST

General | Arc

Infos BDD

Id Jeu Test = 1, Id Arc Precedent = 1,

Capteurs: Camera cycab (CAMERA_CENTRALE), id=1

Infos Images

Extension: pgm

Repertoire:

Du No: Au No:

Echelle initiale: 1.0

Type d'Importation

☒ Inversion Lecture ☒ Avec Propagation ☒ Multiples Arcs

Lancer Importation | Fermer

Decoupe d'un ARC (id = 1)

Capteurs de l'Arc: Camera cycab (CAMERA_CENTRALE), id=1

Partie 1

Id Noeud Debut: 132, 131, 130

Id Noeud Fin: 132, 131, 130

Partie 2

Id Noeud Debut: 132, 131, 130

Id Noeud Fin: 132, 131, 130

Image

Valider | Fermer

FIG. 8: Importing data and cutting edge windows

rate of 15fps. Communications between the embedded PC, the low-level computer which controls the RobuCab and its sensors are performed using the real-time architecture Aroccam [14]. The RobuCab was manually driven along several paths onto our universitary campus.

4.1 Map building

The images acquired along all paths have been first stored. For each of the selected paths, an importation step is performed. This step consists on building an edge, selecting the key images, extracting the 500 relevant image points of each key image and robustly matching two successive images points. The data are stored into the database. Some information about the entire database are detailled in Tab. 1 with the number of data and the memory size onto the disk for some of the main tables. This database contains $3.2{\times}10^6$ data, which results to an amount of 4 255 MB onto the disk. The table **IMAGE** represents 65% of the entire memory size and the table **2DPOINT**

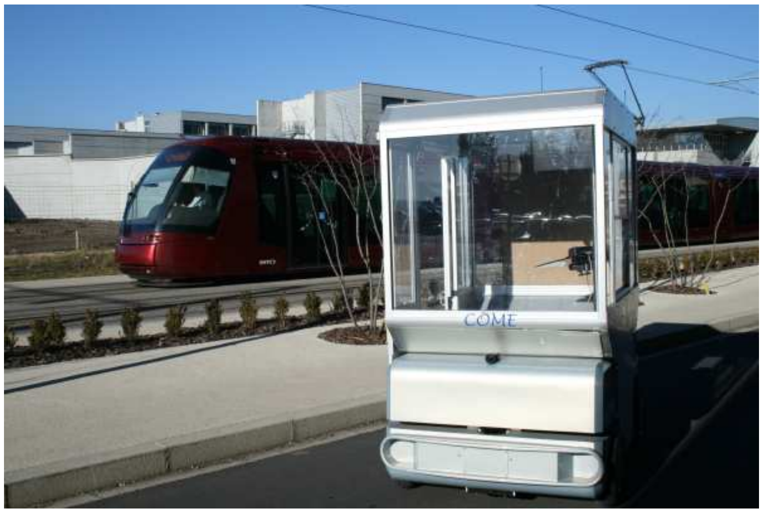


FIG. 9: RobuCab vehicle with the embedded camera.

34%.

Table	Number of data	Memory Size
EDGE	84	2 900 B
NODE	6 067	53 KB
IMAGE	6 067	2 777 MB
2DPOINT	3×10^6	1 473 MB
3DPOINT	230 300	3.7 MB

Table 1: Database contents during the autonomous navigation.

4.2 Visual localization

The vehicle is assumed to be on a known edge. It can be given by the user or by an external sensor such as a GPS. Firstly, the application requires the extraction of the keys of all images, acquired by a given camera, which belong to this edge. For convenience, the images are ordered with respect to the positions of the nodes along the path. Secondly, the localization is a local strategy which requires the loading of the 2D points of the key images. During the importation step, for each key image, 500 2D points have been stored with their descriptors. For each key image, the 500 points are loaded and matched to the current points. The key image with the smallest distance is considered as the current localization in the visual memory. The mean time to load the points of an image is 19 ms by image (mean obtained by loading the points for all images of all edges).
Finally, the full localization process takes approximately 35 ms by key image of the edge.
Note that the time to load a grey-level image of size 800x600 pixels from the database (time to execute the request and to transform it in an image object) takes 16 ms.

4.3 Autonomous navigation

4.3.1 Localization step

The autonomous navigation begins at Start position (refer to Fig. 10). The current image is grabbed (Fig. 11 (a)) and the localization process starts. After 7 seconds, the image is localized into the first edge (Fig. 11 (b)). 337 points are matched. Note that between these two images, illumination conditions have changed as well as the contents (for example, cars disappear, and some objects are different).

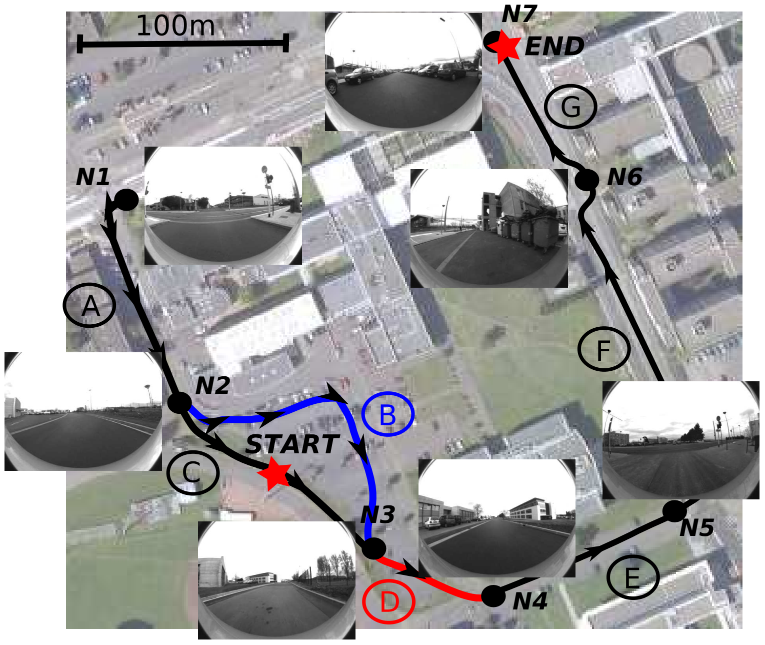


FIG. 10: Some paths of the campus (used in the experimentation) with the memorized trajectories. Edges are represented with their extremity nodes. Images acquired in those nodes are also drawn.

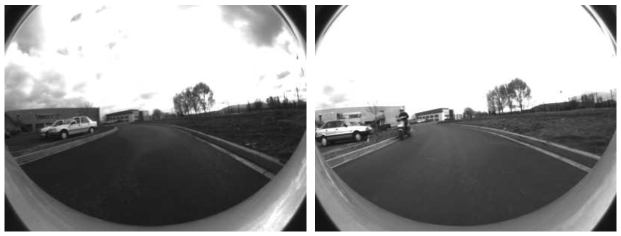


FIG. 11: Left image: current image acquired at Start position. Right image: nearest key image into the edge.

The autonomous navigation consists on following the path $\Psi = C \oplus D \oplus E \oplus F'^1 \oplus F'^2 \oplus G'^1 \oplus G'^2$. Edges with a prime denote edges taken during other days than the first paths. The nodes $N5$ and $N6$ (refer to Fig. 10) are linked by the edge F but also by the succession of F'^1 and F'^2 and the nodes $N6$ and $N7$ are linked by the edge G but also by the succession of G'^1 and G'^2. This path contains 7 edges and 396 key images. The total length of the path is more than 400 m (obtained by odometric measures).

4.3.2 Autonomous navigation

The speed of the vehicle is set to 0.8 m/s. Grey level images, of size 800x600, are acquired at 15fps. At each frame, points are extracted and matched with the desired key image. Robust matching allows us to keep only the valid matchings. From these matchings, the epipolar geometry is computed and the lateral error y and the angular error θ are estimated. The control law is computed and sent to the RobuCab controller. The average computation time is 70 ms for each current image. This time also includes data loading related to the new desired key image when the former is reached. Our vehicle successfully follows the visual path. The errors in the images (the mean of the distances between the matched points) decrease to zero until reaching a key image (refer to Fig. 12). In the figures small crosses denote that a new key image is reached, diamonds that a new edge begins. Some reached images (with the corresponding key images of the memory) are shown in Fig. 14. Note that illumination conditions have changed between the memorization and the autonomous steps as well as the contents but the vision-based navigation strategy succeeds. The

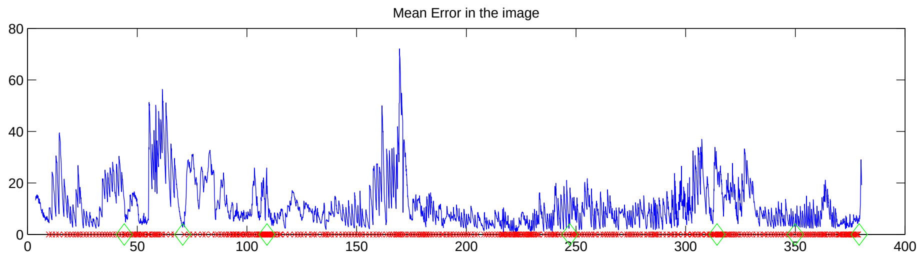


FIG. 12: Errors in the images versus time (s).

lateral and angular errors are also well regulated to zero (refer to Fig. 13).

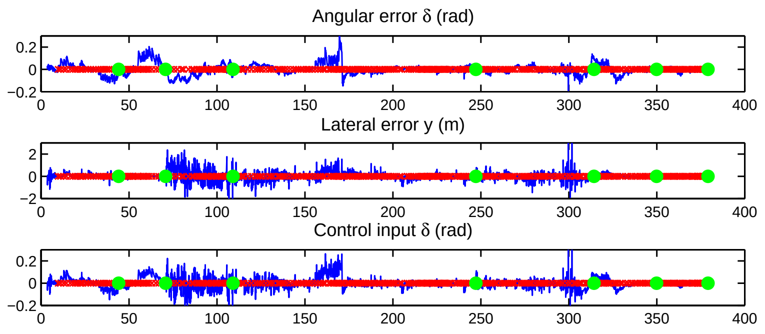
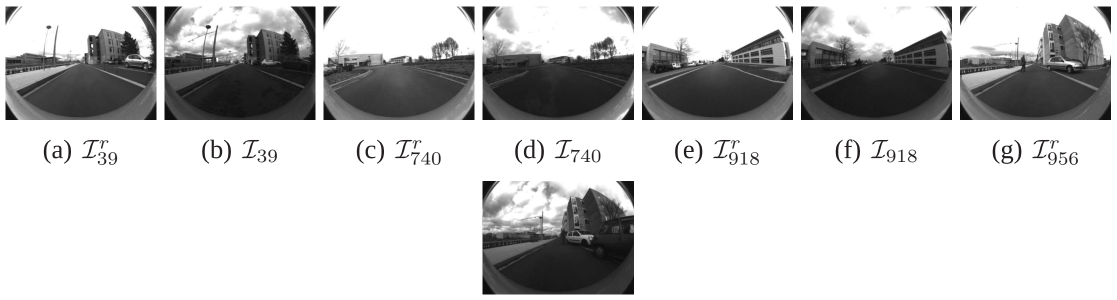


FIG. 13: Angular and lateral errors and control input versus time (s).

5 Conclusion

In this paper, a software for autonomous navigation has been presented. The software platform (SoVin) is more particularly dedicated to visual memory management and vision-based navigation. It allows to achieve navigation tasks in large scale environments using natural landmarks. Preliminary experiments obtained with SoViN have shown promising results. Future works will be devoted



(a) $\mathcal{I}^r_{39}$ (b) $\mathcal{I}_{39}$ (c) $\mathcal{I}^r_{740}$ (d) $\mathcal{I}_{740}$ (e) $\mathcal{I}^r_{918}$ (f) $\mathcal{I}_{918}$ (g) $\mathcal{I}^r_{956}$

(h) $\mathcal{I}_{956}$

FIG. 14: Some of the current images $\mathcal{I}^r_k$ where the key images $\mathcal{I}_k$ have been reached (k is the key of the image in the database).

to intensively experiments the software in various configurations and using different vision-based navigation strategies as the one proposed in [12].

References

[1] J. Courbon, G. Blanc, Y. Mezouar, and P. Martinet, *”Navigation of a non-holonomic mobile robot with a memory of omnidirectional images”*, ICRA 2007 Workshop on ”Planning, perception and navigation for Intelligent Vehicles” (Rome, Italy), 2007.

[2] J. Courbon, L. Lequievre, Y. Mezouar, and L. Eck, *”Navigation of urban vehicle: An efficient visual memory management for large scale environments”*, IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS’08, 2008, submitted.

[3] J. Courbon, Y. Mezouar, L. Eck, and P. Martinet, *”Efficient hierarchical localization method in an omnidirectional images memory”*, IEEE International Conference on Robotics and Automation, ICRA’08 (Pasadena, CA, USA), May 19-23 2008.

[4] G. N. DeSouza and A. C. Kak, *”Vision for mobile robot navigation: A survey”*, IEEE transactions on pattern analysis and machine intelligence **24** (2002), no. 2, 237–267.

[5] A. Diosi, A. Remazeilles, S. Segvic, and F. Chaumette, *”Outdoor visual path following experiments”*, IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS’07 (San Diego, CA, USA), 29 October - 2 November 2007, pp. 4265–4270.

[6] M. Fischler and R. Bolles, *”Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography”*, Comm. of the ACM, vol. 24, 1981, pp. 381–395.

[7] T. Goedemé, T. Tuytelaars, G. Vanacker, M. Nuttin, L. Van Gool, and L. Van Gool, *”Feature based omnidirectional sparse visual path following”*, IEEE/RSJ International Conference on Intelligent Robots and Systems (Edmonton, Canada), August 2005, pp. 1806–1811.

[8] C. Harris and M. Stephens, *”A combined corner and edge detector”*, Alvey Conference, 1988, pp. 189–192.

[9] D.G. Lowe, *"Distinctive image features from scale-invariant keypoints"*, Int. Journal of Computer Vision, vol. 60, 2004, pp. 91–110.

[10] Y. Matsumoto, M. Inaba, and H. Inoue, *"Visual navigation using view-sequenced route representation"*, IEEE International Conference on Robotics and Automation, ICRA'96 (Minneapolis, Minnesota, USA), vol. 1, April 1996, pp. 83–88.

[11] D. Nistér, *"An efficient solution to the five-point relative pose problem"*, Transactions on Pattern Analysis and Machine Intelligence **26** (2004), no. 6, 756–770.

[12] E. Royer, M. Lhuillier, M. Dhome, and J.-M. Lavest, *"Monocular vision for mobile robot localization and autonomous navigation"*, International Journal of Computer Vision, special joint issue on vision and robotics **74** (2007), 237–260.

[13] J. Shi and C. Tomasi, *"Good features to track"*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94) (Seattle), June 1994.

[14] C Tessier, C. Cariou, C. Debain, F. Chausse, R. Chapuis, and C. Rousset, *"A real-time, multi-sensor architecture for fusion of delayed observations: Application to vehicle localisation"*, 9th International IEEE Conference on Intelligent Transportation Systems (Toronto, Canada), September 17-20 2006, pp. 1316–1321.

[15] B. Thuilot, J. Bom, F. Marmoiton, and P. Martinet, *"Accurate automatic guidance of an urban electric vehicle relying on a kinematic GPS sensor"*, 5th IFAC Symposium on Intelligent Autonomous Vehicles, IAV'04 (Instituto Superior Técnico, Lisbon, Portugal), July 5-7th 2004.

[16] P.H.S. Torr and A. Zisserman, *"Mlesac: a new robust estimator with application to estimating image geometry"*, Computer Vision and Image Understanding. Special issue on robusst statistical techniques in image understanding **78** (2000), 138–156.

[17] S. Šegvić, A. Remazeilles, A. Diosi, and F. Chaumette, *"Large scale vision based navigation without an accurate global reconstruction"*, IEEE International Conference on Computer Vision and Pattern Recognition, CVPR'07 (Minneapolis, Minnesota, USA), jun 2007, pp. 1–8.