Tesis Doctoral

Sistema de navegación monocular para robots móviles en ambientes interiores/exteriores

De Cristóforis, Pablo

2013

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Sistema de navegación monocular para robots móviles en ambientes interiores/exteriores

Tesis presentada para obtener el título de Doctor
de la Universidad de Buenos Aires
en el área Ciencias de la Computación

Pablo De Cristóforis

Directora: Dra. Marta Mejail

Lugar de Trabajo: Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires.

Praga, 2013.

For those who struggle to defend Public Education and Research.

Sistema de navegación monocular para robots móviles en ambientes interiores/exteriores

Uno de los desafíos actuales de la robótica móvil es alcanzar el mayor grado de autonomía, es decir, lograr que un robot desarrolle sus tareas sin la necesidad de un operador humano. El objetivo principal de este trabajo es el desarrollo de un nuevo sistema de navegación autónomo basado en visión para robot móviles en entornos interiores/exteriores. El sistema propuesto utiliza sólo una cámara y sensores de odometría, no depende de ningún sistema de localización externo o infraestructura similar. Además, es capaz de tratar con variaciones en el ambiente (por ejemplo, cambios de iluminación o estaciones del año) y satisface las restricciones para guiar al robot en tiempo real.

Para alcanzar el objetivo de este trabajo, se propone un enfoque híbrido que hace uso de dos técnicas de navegación visual: una basada en segmentación de imágenes y otra basada en marcas visuales. Para representar el ambiente se construye un mapa topológico que puede ser interpretado como un grafo, donde las aristas corresponden a caminos navegables y los nodos a espacios abiertos. Para recorrer los caminos (aristas) se desarrolló un método original basado en segmentación y para navegar por los espacios abiertos (nodos) se realizó una mejora y adaptación de un método basado en marcas visuales. Se evaluaron diversos algoritmos de extracción de características distintivas de las imágenes para determinar cuál representa la mejor solución para el caso de la navegación basada en marcas visuales, en términos de performance y repetibilidad.

El sistema desarrollado es robusto y no requiere de la calibración de los sensores. La convergencia del método de navegación se ha demostrado tanto desde el punto de vista teórico como práctico. Su complejidad computacional es independiente del tamaño del entorno. Para validar el método realizamos experiencias tanto con sets de datos como con el robot móvil ExaBot, que se presenta como parte de este trabajo. Los resultados obtenidos demuestran la viabilidad del enfoque híbrido para abordar el problema de la navegación basa en visión en entornos complejos interiores/exteriores.

Palabras clave: robots móviles, navegación autónoma basada en visión, segmentación de imágenes, características de imágenes.

Vision-based mobile robot system for monocular navigation in indoor/outdoor environments

One of the current challenges of mobile robotics is to achieve complete autonomy, i.e. to develop a robot that can carry out its tasks without the need of a human operator. The main goal of this work is to develop a new vision-based mobile robot system for autonomous navigation in indoor/outdoor environments. The proposed system uses only a camera and odometry sensors, it does not rely on any external localization system or other similar infrastructure. Moreover, it can deal with real environmental changing conditions (illumination, seasons) and satisfies motion control constraints to guide the robot in real time.

To achieve the goal of this work, a hybrid method is proposed that uses both segmentation-based and landmark-base navigation techniques. To represent the environment, a topological map is built. This map can be interpreted as a graph, where the edges represent navigable paths and the nodes open areas. A novel segmentation-based navigation method is presented to follow paths (edges) and a modified landmark-based navigation method is used to traverse open areas (nodes). A variety of image features extraction algorithms were evaluated to conclude which one is the best solution for landmark-based navigation in terms of performance and repeatability.

The developed system is robust and does not require sensor calibration. The convergence of the navigation method was proved from theoretical and practical viewpoints. Its computational complexity is independent of the environment size. To validate the method we perform experiments both with data sets and with the mobile robot ExaBot, which is presented as part of this work. The results demonstrate the feasibility of the hybrid approach to address the problem of vision based navigation in indoor/outdoor environments.

Keywords: mobile robots, autonomous vision-based navigation, image segmentation, image features.

Acknowledgement

First of all, I would like to express thanks to my supervisor, Marta Mejail, for maintaining an academical environment at our robotics laboratory which allowed to pursue this thesis topic. She is a person I really respect, who taughed me a lot, inspired me and supported me during my studies. I also would like to express my thanks to all members of the Laboratory of Robotics and Embedded Systems, in particular to Sol Pedre and Javier Caccavelli, with whom the ExaBot robot came true, to Matías Nitsche for his work with the algorithms implementation, to Facundo Pessacg for his technical assistance with the hardware issues, and to Taihú Pire for his help with the experiments.

Moreover, I would like thank all members of the Intelligent and Mobile Robotics group of the Czech Technical University in Prague, where I spent a very productive stays during these years. I would like to thank particularly Tomáš Krajník, for his valuable remarks on my research. His ideas were essential to develop the navigation methods presented in this Thesis.

I am very grateful to my parents Ana and Héctor, my sisters Nadia and Mariel, and my little niece Cloé. Moreover, I am very gratful to all close friends and colleagues from the Computer Science Department of the FCEN-UBA, for their support during my studies. And finally, and the most important for me, I want to thank my lovely Renata, for her constant support and endless patience. She was my beacon of light in the darkest moments of my PhD studies that allowed me to navigate till the end of this work.

Contents

Chapter 1

Introduction

This Chapter gives a perspective of the presented work related to the field of robotics. It starts with the history of robots, then presents some of today's applications and finally puts forward the aim and structure of this Thesis.

1.1 Looking backward at the Robot

The dream of creating an artificial creature has been part of humanity through centuries. It is possible to find it in the Jewish folklore in the form of a *Golem*, an anthropomorphic being, created from inanimate material. One of the goals in Alchemy was to create a *Homunculus*, a word introduced in medieval writing derived from Latin (diminutive of *homo*) which means little man. During the Islamic Empire, the arabic word *Takwin* refers to the artificial creation of life in the laboratory. From medieval stories through Mary Shelley's gothic novel *Frankenstein*, the subject is the same: a mad scientist works obsessively to create an artificial being, trying to be God, ignoring the dark forebodings and danger of the consequences of his research.

However, the word *Robot* did not appear until the beginning of the XX century and it is no coincidence that this happened when the Industrial Revolution had already spread across Europe. It was in 1921, during the presentation of the play R.U.R (Rossum's Universal Robots) in the National Theater of Prague. The word, suggested to Karel Čapek by his brother Josef, is a neologism made from the old Czech word of Slavonic origin *Robota*, which means compulsory labor, drudgery, or hard work. The meaning of the word robot is broader than the originally intended interpretation as *Labors* (Labori), by Karel Čapek. While *labor* are men reduced to work, *Robot* etymologically refers to an orphan. That is why the word *Robot* better resonates with the notion of an artificial man, one which was not naturally born. Josef and Karel Čapek coined the word robot for the first time in the play R.U.R. The play

conceives a future time in which all workers will be automated. Their ultimate revolt, that happens when they acquire souls, and the ensuing catastrophe result in an exciting and vivid theatrical experience. The play gained immediate popularity and was staged all around the world, and thus the word *Robot* was made known everywhere [1].

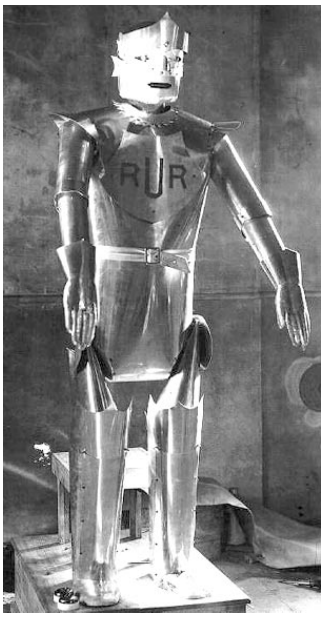


Figure 1.1: Man-shaped robot representation which appeared in an adaption of R.U.R. (Rossum's Universal Robots) play in 1930.

Looking back at the beginning of the 20th century we can see, that the robot became a popular figure working in service of the modern myth of technological progress. Robots colonized pages of science-fiction novels and short stories very soon. The most influential neologism derived from the word robot is *Robotics*, coined by Issac Asimov. He used this word, as well as the well known *Three Laws of Robotics*, for the first time in his short story *Runaround*, published in pulp magazine *Astounding Science Fiction* in 1942 [2]. The author, who calls himself "father of the modern robot stories" inspired many scientists and engineers working in robotics laboratories in their effort to construct a humanoid robot.

The next step, which is of more interest for this Thesis, comes at the end of the 1960's when the first general-purpose mobile robots controlled primarily by programs which reasoned were built. The robot Shakey, developed at the Artificial Intelligence Center of Stanford Research Institute (now called SRI International), was the first one. Between 1966 and 1972 the Shakey project faced basic problems of mobile robotics such as object recognition, obstacle avoidance, world model building in structured (very simple) environments and symbolic planning by using video processing [3]. Fantasies and dreams started

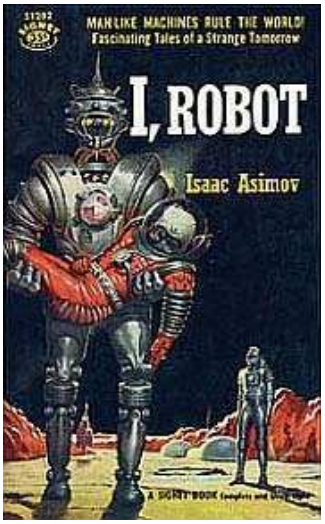


Figure 1.2: The neologism *Robotics* and the original *Three Laws of Robotics* were coined by Isaac Asimov in his 1942 short story *Runaround*. Eventually it became only one of several similar stories published under the common name *I, Robot*, first published by Gnome Press in 1950.

to be envisioned by scientists. The success of artificial intelligence research in the 1960s and 1970s had inspired expectations in the domain of mobile robotics. After some years of research it was clear that it was comparatively easy to make computers to exhibit adult-level performance in solving some problems on intelligence tests, but it was awfully difficult to endow them with the skills of a one-year-old child when it comes to perception and mobility.

Hans Moravec, one of the pioneers of mobile robotics in the 1970's, was exactly of this opinion when he wrote his book: *Mind Children, The Future of Robot and Human Intelligence* in 1988 [4]. However, even by knowing this, he expected human-like performance in mobile robotics by the end of the last millennium. Now, a few years of the new millennium have passed and we know that Moravec made a mistake because there is a lot of work to be done before achieving this goal. But we have made progress: nowadays robots are starting to move from prototypes in the laboratories into a reality in factories, agricultural fields, households, streets, schools and even in other planets. Scientists and engineers working in robotics still have the ambition to fulfill the ancient dream: to build an artificial or mechanical creature: a Robot.

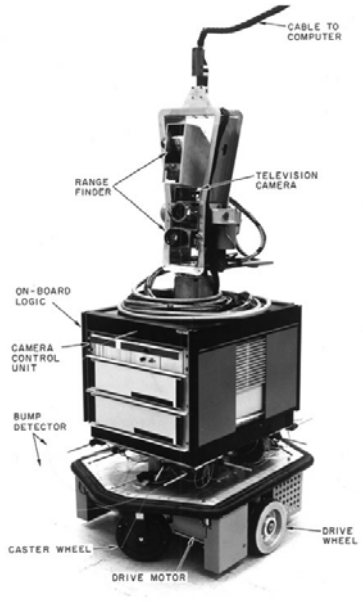


Figure 1.3: Shakey was the first general-purpose mobile robot to be able to reason about its own actions. It was developed at the Artificial Intelligence Center of Stanford Research Institute (now called SRI International) between 1966 and 1972.

1.2 Applications of Mobile Robots

Mobile robotics has become one of the most interesting areas in field of robotics. The development of new mobile robot prototypes has made significant progress in recent decades, which has allowed their use in different environments, for a variety of purposes.

In factories, mobile robots are used for material handling. Given the large flow of materials in industrial production lines, Automated Guided Vehicles (AGVs) are helpful components. The main task of AGVs in industrial environments is usually to transport materials from one workbench to another, which is a tedious and sometimes dangerous task for human operators. The AGV's autonomy is based on a set of automatic processes, such as perception, path planning and path tracking, that the industrial vehicle must perform to accomplish the handling task. On their way they have to avoid obstacles and at the final docking point they have to approach it with high accuracy. Industrial production environments are usually structured and well known. Thus, maps exhibiting walls, doors, rails on the floor, machines and other characteristic fixed landmarks can be used for navigation [5, 6, 7].

In agriculture, mobile robots can be used to reduce the environmental pollution caused by excessive employment of herbicides. A potential way to

reduce chemicals is to apply other methods, e.g. mechanical weed control between plants in the seed line. This task can be carried out by vision-based mobile robot systems [8]. Another example is the case of autonomous tractors, developed to help in row crop husbandry for precision guidance of an agricultural cultivator or forage harvester [9]. For large fields, cooperation of multiple autonomous and human-operated tractors has been demonstrated, for example, in peat moss harvesting operations. The behavior and actions of these autonomous tractors were designed to mimic manual harvest operations while maintaining a safe operating environment [10].

(a) Hortibot (b) Modified OMG 808-FS

Figure 1.4: Mobile robots in factories and agriculture: (a) Hortibot is a prototype agricultural robot that uses autonomous navigation, cameras and a variety of tools for weed control. (b) Modified OMG 808-FS commercial forklift truck in an industrial warehouse.

Mobile robots have been used for exploration as well. Perhaps the most famous mobile robots are those used for space exploration. Nowadays, using mobile robots is the only viable option for exploring distant planets. These robots are able to explore large areas, can handle the extreme conditions of outer space, carry out scientific experiments on-board and, unlike human astronauts, they do not need to be returned to Earth. The first mobile robot which landed and explored a celestial body was the soviet Lunokhod 1, in 1970. It was a rover that carried several videocameras used for its navigation on the Moon surface. However, the Lunokhod was teleoperated from Earth and it was not designed for autonomous operation. The first autonomous mobile robot that explored another planet was the Prop-M rover, which landed on Mars in 1971 as a part of Mars 3, an unmanned space probe of the Soviet Mars program. It looked like a small box with a small ledge in the middle. The devices were supposed to move over the surface using two skis. Two thin bars at the front were the sensors to detect obstacles. The rover could determine by itself on which side was the obstacle to retreat from it and try to get around. Autonomy is necessary for the Mars mobile devices, since a signal from Earth

to Mars takes between 4 and 20 minutes, which is too long for a mobile robot control. Twenty five years later, the USA managed to place its first rover on Mars. It was Sojourner from NASA, which landed in 1997 [11]. The Sojourner vehicle was partially autonomous, the computationally demanding tasks were performed at the Earth control center, where human operators used a 3D environment model recovered from the rover stereo cameras to set up a series of waypoints. After that, the rover moved between these waypoints with a simple reactive behaviour using structured light to detect obstacles. A new rover Curiosity is exploring Mars at the time this Thesis is being written [12].

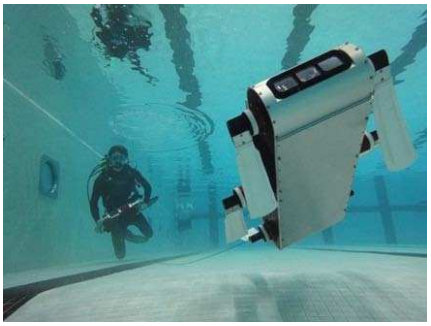
However, outer space is not the only place where mobile robotics can be applied for autonomous exploration. Oceanographic and undersea exploration can be performed by autonomous underwater vehicles (AUV's). Equipped with the necessary sensor technology, these robots are able to inspect port waters or venture down to the ocean floor in search for natural resource deposits [13]. Unmanned Aerial Vehicles (UAV's), commonly known as drones, are aerial robots that can also be used for exploring hostile or dangerous environments for human beings, for example after a natural disaster like an earthquake [14], or after a nuclear and radiation accident [15]. Moreover, UAV's can be used for surveillance, for example in National Parks to overfly large-scale area and prevent the hunt of endangered species or quickly detect forest fires [16].

Roads and urban streets are another environment where the autonomous mobile robots have been tested. This is the case of the 'intelligent' cars. The Defense Advanced Research Projects Agency (DARPA) Grand Challenge is a prize competition originated on the United States, for the goal of achieving driverless vehicles [17]. The initial DARPA Grand Challenge was created to spur the development of technologies needed to achieve the first fully autonomous ground vehicles capable of completing a substantial off-road course within a limited time. Although unsuccessful in the first years, in 2005, a team led by a known mobile robotics researcher, Dr. Sebastian Thrun, created the robotic vehicle known as Stanley at Stanford University, which completed the course and won the DARPA Grand Challenge [18]. The third competition of the DARPA, known as the Urban Challenge, extended the initial challenge to autonomous operation in a mock urban environment. This time, the autonomous cars had to move in an urban environment with simulated road traffic including pedestrians [19]. Following his successes, the Google company hired Thrun and started their own autonomous car project. In August 2012, the Google team announced that they had completed over 300,000 autonomous-driving miles, accident-free. Three USA states (Nevada, Florida and California) have passed laws permitting driverless cars on September 2012. Some ideas of Thrun's autonomous car are analyzed and implemented in this Thesis.

As the prices of sensors, computational hardware and electronic components decrease, new horizons for mobile robots in the domestic sphere have

(a) Mars Rover Curiosity

(b) Stanley autonomous vehicle



(c) Aqua2 under water robot

(d) Parrot AR.Drone

Figure 1.5: Mobile robots in the air, under water, on the road and on Mars: (a) Curiosity developed by NASA is now exploring Mars surface, (b) Stanley was an autonomous vehicle that was able to navigate 300 km through desert terrain in less than 10 hours, (c) Aqua2 is a under water robot used for projects ranging from reef monitoring to aquaculture inspection. (d) The Parrot AR.Drone is a flying quadrotor helicopter commonly used for testing autonomous navigations in UAVs.

opened up. Many household maintenance chores do not require high cognitive capabilities and can be performed with reactive algorithms. Brook's subsumption architecture [20] gave a new reactive paradigm to control robots using a behavior-based approach. In this way, small mobile robots, albeit with limited sensory and computational power, are able to perform simple tasks with satisfactory efficiency. The most notorious examples are the Roomba floor cleaning robots [21] and the RoboMow robotic lawn mower [22]. These robots are equipped with cheap infrared sensors, magnetic sensors and touch sensors, which allow to avoid obstacles and to search for a charging station. Other mobile robots have appeared at homes as pets. This is the case of AIBO (Artificial Intelligence roBOt), an iconic series of dog-like robotic pets designed and manufactured by Sony between 1999 and 2006 [23]. In recent years Genibo, produced by Dasarobot took the place of his predecessor [24].

Educational robotics proposes the use of mobile robots as a teaching resource that enables inexperienced students to approach topics in fields related

and unrelated to robotics. One of its aims is to aid school students in building their own representations and concepts of science and technology through the construction, handling and control of robotic environments. Educational robotics is a growing field, with LEGO Mindstrom being the most popular kit for outreach activities [25]. For teaching topics in robotics at Universities, different platforms have been developed in recent years for mobile robotics, artificial intelligence, control engineering, and related domains [26]. The mobile robot ExaBot, presented in this Thesis, was also designed and successfully tested for this goal [27].

(a) Roomba floor cleaning

(b) Aibo dog-like robot

(c) Smart Pal V service robot

(d) LEGO Mindstorms

Figure 1.6: Mobile Robots for edutainment and service: (a) Roomba-560 is an autonomous floor cleaning machine, (b) Sony Aibo dog-like robot was an iconic robotic pet between 1999 and 2006, (c) Smart Pal V is a service robot developed by Yaskawa assists human beings, typically by performing household chores. (d) The LEGO Mindstorms series of kits contain software and hardware to create small, customizable and programmable robots.

Service robots can be particularly useful for disabled or elderly people. Since these individuals can be limited in their mobility, a service robot can encourage independent living. A robotic wheelchair can provide users with

driving assistance, taking over low-level navigation to allow its user to travel efficiently and with greater ease [28]. Service robots can also aid with physical tasks, and they can facilitate many cognitive and social services. For instance, they might give the person a way to communicate with friends or relatives, link the person to his or her doctor or give the person daily reminders [29]. Other service mobile robots have also been developed for guiding blind people [30].

There is also a variety of mobile robot systems for military aims, however these applications have been intentionally excluded from this thesis.

1.3 About this work

The main goal of this Thesis is to develop a new vision-based mobile robot system for monocular navigation in indoor/outdoor environments. The proposed method should reliably guide an autonomous robot along a given path. It should use only a camera and odometry sensors and it should not rely on any external localization system or other similar infrastructure. Moreover, it should be able to deal with real environmental changing conditions (illumination, seasons) and satisfy motion control constraints to guide the robot in real time.

To achieve the goal of this Thesis, a hybrid method is proposed that uses both segmentation-based and landmark-base navigation techniques. A topological map is built to represent the environment. This map can be interpreted as a graph, where the edges correspond to navigable paths and the nodes correspond to open areas. A novel segmentation-based navigation method is presented to follow paths (edges) and a landmark-based navigation method is used to traverse open areas (nodes). A variety of image feature extraction algorithms were evaluated to conclude which one represent the best solution for landmark-based navigation in terms of performance and repeatability. The proposed method is intended to be applicable to small low-cost mobile robots and off the shelf cameras. In this Thesis, to test the method, a new low-cost mobile robot called ExaBot was also developed. This mobile robot provides a suitable platform able to adapt to different experiments.

In the next chapters, the aforementioned issues will be addressed in detail:

- First, the thesis reviews the state of the art in mobile robot navigation in Chapter 2.
- Chapter 3 presents the ExaBot: a new mobile robot developed at the Laboratory of Robotics, of the Department of Computer Science, Faculty of Exact and Natural Sciences, University of Buenos Aires (FCEN-UBA). The ExaBot is then used for experimental trials in Chapters 4, 5 and 7.

- In Chapter 4, a novel method for autonomously drive mobile robots through outdoors paths is detailed. This method is based on segmenting the images captured with the camera and classifying each region to infer a contour of navigable space.

- In Chapter 5, a known landmark based visual navigation method is enhanced and described for the ExaBot. This method uses the teach-and-replay paradigm for navigation in indoor and outdoor unstructured environments.

- In Chapter 6, the Thesis evaluates a variety of image features detector-descriptor schemes for visual navigation. SIFT, SURF, STAR, BRIEF and BRISK methods are tested using a long-term robot navigation data set. The goal of this Chapter is to conclude which image feature detector-descriptor scheme is best for visual navigation in terms of performance, robustness and repeatability.

- In Chapter 7, an hybrid approach for monocular robot navigation in indoor/outdoor environments is presented. This approach uses both segmentation and features from images for visual navigation, and a topological map to reliably guide an autonomous robot along an indoor/outdoor given path.

- The last Chapter 8 concludes this Thesis and summarizes the whole work.

Chapter 2

Methods of Mobile Robotics

This Chapter addresses fundamental questions concerning mobile robotics, focusing on autonomous navigation.

2.1 Introduction

One of the current challenges of mobile robotics is to achieve complete autonomy, i.e. to develop a robot that can carry out its tasks without the need of a human operator. The ability to navigate by itself in its environment is fundamental for an autonomous mobile robotic system. Navigation can be roughly described as the process of moving safely along a path between a starting point and a final point. In order to navigate in the real world, a mobile robot needs to take decisions and execute actions that maximize the chance to reach the desired destination. Moreover, during this process it has to avoid collisions and also detect those portions of the world that are forbidden, dangerous or impossible to traverse.

The general problem of mobile robot navigation can be summarized by three questions: Where am I? Where am I going? How do I get there? [31] These three questions are respectively answered by localization, mapping and motion planning. In the context of mobile robotics, localization means determination of the robot position in the environment [32]. Mapping addresses the problem of building models or representations of the environment from data acquired by robot sensors [33]. And the process of determining the path to the desired destination and generating inputs for robot actuators to follow the path is referred to as motion planning. [34].

Although one can address these problems separately, they are closely related and especially localization and mapping are often tackled together. For example, the robot can incrementally add new information to a map and use this map to estimate its position. This is called Simultaneous Localization and

Mapping (SLAM), a method used to build up a map within an unknown environment (without *a priori* knowledge) while at the same time keeping track of robot current location [35] [36].

In the next Sections of this Chapter each of these aforementioned main issues are explained in detail, and the state of the art in mobile robot methods is reviewed. Finally, visual-based navigation approach is presented, as it relates directly to what we draw on in this Thesis.

2.2 Localization

Localization is a problem of pose (position and orientation) estimation of a vehicle in either absolute or relative frame of reference. One can distinguish between relative (or local) localization, where current position of the vehicle is computed using a previously known position and data provided by on-board sensors, and absolute (or global) localization, where the position is obtained using information from exteroceptive sensors, like beacons or landmarks.

A special problem happens when the robot is suddenly moved to an arbitrary location. This situation is commonly referred to as the "kidnapped robot problem", where the robot has no *a priori* knowledge of its location. The kidnapped robot problem creates significant issues to the robot localization system, so it is commonly used to test a robot's ability to recover from catastrophic localization failures.

Relative localization is also referred to as dead reckoning, a term originating from nautical navigation, where a ship had to estimate its speed relative to 'dead' water. Relative localization is used with a high sampling rate in order to maintain the robot pose up-to-date, whereas absolute localization is applied periodically with a lower sampling rate to correct relative positioning misalignments [37].

2.2.1 Relative localization

The most popular relative localization method in mobile robotics is odometry. This method calculates the robot position and orientation from the signals of its actuators, either wheels [38] or legs [39]. Most frequently, the wheeled robots are equipped with incremental rotary (or shaft) encoder sensors attached to wheel axes. The idea of this technique is to translate the wheel speed (measured by the encoder) in a linear displacement of the robot relative to the ground. It is well known that odometry may provide good short-term estimation in linear displacement since it allows very high sampling rates, it is also computationally inexpensive and easy to implement. However, the funda-

mental idea of the odometry information integration is incremental movement over time, which leads to the accumulation of errors which inevitably increases proportionally with the distance traveled by the robot.

Odometry errors are of two types: systematic errors, dependent on structural characteristics of the robot (for example, the diameters of the wheels are not equal), and non-systematic errors that do not depend on the robot but on the environment. Non-systematic errors break the assumption that the encoder pulses can be translated into a linear displacement relative the ground (for example, when wheels slip on the floor). Systematic errors can be treated and taken into account when computing the position and orientation of the robot, but non-systematic errors are unpredictable and unmanageable [40].

Other dead reckoning methods rely on the use of inertial measurement units [41] and gyroscopes [42]. To measure the true acceleration of a vehicle, one has to take into account the gravity of Earth as well as centrifugal and Coriolis forces. Since the speed is measured indirectly by integration of acceleration measurements, and therefore, position is computed by double integration, then the problem of error accumulation is even more critical. These methods are applicable to vehicles which are not in contact with a solid surface and have been successfully used in UAV's robots and space flight domain.

In recent years, new dead reckoning methods have been developed based on visual input. Using sequential camera images it is possible to compute the relative motion between two frames and thus get an accurate estimation of the robot pose. This idea leads to a new relative location technique known as visual odometry, term coined in [43]. Most of visual odometry methods consist in extraction of features from consecutive images, then matching the features using their descriptors in the second image, effectively tracking them for the estimation of the egomotion of the camera (and of course, also the robot) by using this information. Some visual odometry algorithms have made use of monocular cameras [44], omnidirectional cameras [45] and stereo cameras, either as the sole sensor [46] or in combination with inertial measurements [47].

The main disadvantage of relative localization or dead reckoning methods rests on the fact that the position error grows over time. Although error accumulation can be slowed down considerably by using precise sensors and careful sensor calibration, measurement errors get integrated over time making dead reckoning unsuitable for long-term localization. Even if the sensor noise were completely eliminated, a small error in initial position angle estimation would cause loss of position precision. Nevertheless, dead reckoning is a popular technique for quick estimation of robot position as a part of more complex localization systems that include other sensors [48] or when the estimation of the position of the robot has to be used only for a short term during autonomous navigation [49].

2.2.2 Global localization

To remove the problem of cumulative errors in dead reckoning, the robot has to interpret signals from its exteroceptors and compare them to an *a priory* known map. Map-based localization methods are always dependent on map representation.

Another approach for global localization consists in the detection of salient, well distinguishable localized features in sensor data, called landmarks. The robot has to identify landmarks and measure their bearings relative to each other. For identifying landmarks, similar features already stored in a landmark map have to be matched with currently measured features using sensor information. [50]. Such sensor information is generally uncertain and contains noise. Given the positions of landmarks on a 2D or 3D map of the environment and the noisy measurements of their bearings relative to each other, the robot position is estimated with respect to the map of the environment by methods originating from geometry. The major disadvantage of this approach is that reliable and robust recognition and matching of salient features in the environment can be computationally expensive and sometimes impossible due to sensor noise. Commonly, landmark-based localization is performed simultaneously with landmark-based mapping leading to SLAM (see Section 2.4). Typically, the landmarks are identified from camera images, but landmark extraction is also possible from laser [51], sonar [52] or radar data [53].

A practical way to provide sensor data with well detectable and distinguishable landmarks is creating them by designing special, salient objects and placing them in the environment. This is done by the use of so-called beacons. A beacon is an artificial object, which is added to the environment to provide aid for localization. A typical beacon-based system is composed of several beacons on known positions. A vehicle carries a detection system, which provides angles and/or ranges to individual beacons and computes its position by means of triangulation [54] or trilateration [55]. The advantage of triangulation, i.e. determination of robot position from angles to individual beacons, over trilateration, i.e. estimation of robot position from beacon distances, is that triangulation provides not only robot position, but also its orientation.

One can distinguish beacons as either passive or active. Unlike passive beacons, active beacons emit signals which can be received by vehicles. The most notorious active beacon examples are lighthouses and the Global Positioning System (GPS) [56]. The GPS is nowadays the most popular localization system, mainly because the GPS receivers are commercially available and affordable. It consists of several active beacons that are actually satellites orbiting the Earth and transmitting signals. The receiver computes its position based on distances from individual satellites with known positions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. The usual GPS-based localization precision in

areas with direct sky visibility and good weather conditions is about 3 meters. However, the GPS signal is easily absorbed and reflected by water. As a result, the GPS precision deteriorates when used in canyons, woods, heavy foliage or between buildings. One can overcome the temporary decrease of localization precision by combining GPS with dead-reckoning localization systems. Alternatively, if the GPS receiver has data about its surrounding environment, it can use this information to reject reflected signals and improve the localization precision [57].

2.3 Mapping

Mapping addresses the problem of acquiring models of the surrounding environment. In mobile robotics, mapping is a process during which the robot builds its own representation about the operating environment, i.e. a map. The type and quality of the map typically depends on the navigation task and sensors. The most common use of a map is to help motion planning and localization of a single robotic agent. However, it can be used for sharing environment information between several robots in multi-agents systems. When the navigation task is focused on exploration, building a map is the goal for the purpose of creating a precise model of the environment.

It can be distinguished between metric and topological maps. Metric maps are built from metric sensor data and describe geometric properties of the environment in a fixed coordinate frame. These are easy to create, and suitable for tasks in small environments (path planning, collision avoidance, position correction). However, their use in large environments is questionable due to space and computational complexity. Examples of metric maps are occupancy grids, geometrical and landmark maps. By contrast, topological maps are based on recording topological or spacial relations between observed environment features rather than their absolute positions. The resulting representation takes the form of a graph, where the nodes represent the observed features or salient places in the environment and the edges represent their relations.

2.3.1 Metric maps

Grids maps

One of the oldest and most popular mapping approaches in mobile robotics is the use of grid maps. In this approach, the environment is discretized into equally sized cells. Each cell represents the area of the environment it covers and has an assigned state, which describes some physical property of this area. In the most common form, each cell contains a probability of being occupied

by an object in the environment, which leads to so-called occupancy grids. Occupancy grid mapping refers to a family of computer algorithms in probabilistic robotics for mobile robots which address the problem of generating maps from noisy and uncertain sensor measurement data, with the assumption that the robot pose is known. The basic idea of the occupancy grid is to represent a map of the environment as an evenly spaced field of binary random variables each representing the presence of an obstacle at that location in the environment. Occupancy grid algorithms compute approximate posterior estimates for these random variables. It is assumed that the occupancy probabilities are independent, and therefore, the impact of sensor measurements on the occupancy probability can be computed for each cell separately [58].

Each sensor measurement can change the occupancy probability of several cells depending on the sensor model, which characterizes the influence of a particular sensor reading to the occupancy of the surrounding cells. A typical example is a rangefinder measurement, which increases the occupancy probability of the cells with the distance equal to the sensor reading while decreasing the occupancy probability of the cells through which the ray has travelled. Since the cells in an occupancy grid can be accessed directly, the update of the grid after a sensor measurement depends only on the sensor model and not on the grid size. Several sensors can be used to update the grid simultaneously, making occupancy grid a simple and elegant method for sensor fusion [59].

There are other approaches for grid-based mapping. For example, if the cells contain local terrain height instead of only occupancy probability, the map is called an elevation grid [60]. Another possibility to model terrain is to store local terrain variance in each cell [61].

The disadvantage of occupancy grid algorithms is their memory inefficiency. Most of the environment is empty space and therefore most cells of a typical occupancy grid are unused. The memory requirement is particularly problematic with threedimensional grids of large environments. However, this problem can be partially solved by using octree spatial representation [62].

Geometrical maps

Geometrical maps attempt to overcome the disadvantage of memory inefficiency and represent only important areas of the environment. Moreover, they do not discretize the environment and therefore are more precise compared to occupancy grids. The geometrical maps model the environment by a set of geometrical primitives, typically lines [63] or polygons [64] in the two dimensional case and planes in the three dimensional case [65].

Geometrical maps are suitable for localization [66] as well as for motion planning [67]. Though memory efficient, these maps are not easy to build

due to sensor noise and localization uncertainty. Moreover, outdoor terrain is usually too complex to be represented by a few geometrical primitives.

Sensor maps

A sensor map is created by recording and storing the sensory measurements without further processing. An example of a sensor map is a point cloud, where range measurements are stored in a global reference frame. Despite the fact, that such a map is not very useful by itself, sensor maps preserve all the measured data and are usually created with the goal of processing the gathered data in the future. For example, the Robotics Data Set Repository (Radish for short) provides a collection of standard robotics data sets in raw format [68].

Landmark maps

Landmarks are distinct features of the environment that a robot can recognize from its sensory input. There are two types of landmarks: artificial and natural. Natural landmarks are those objects or features that are already in the environment whereas that artificial landmarks are specially designed objects or markers that need to be placed in the environment with the sole purpose of enabling robot navigation. Landmarks maps are built by detecting salient features in the environment and computing their relative position to the robot. The primary purpose of a landmark map is localization, so in many works these two problems are tackled together, as we discussed in Section 2.2.2. This is very common in vision-based mobile robot localization and mapping (SLAM) methods, which use image features as natural landmarks [69]. However, landmarks can be used for motion planning as well [70]. An important problem is the extraction of the salient landmarks from the sensory data, which might be computationally expensive. During the map building as well as during localization, the perceived landmarks have to be associated with the landmarks stored in the map. Unless an intelligent landmark preselection is performed, the computational expense of the association process increases with growing map size.

2.3.2 Topological maps

Topological maps represent the world as a graph, i.e. a network of nodes and edges where the nodes are distinctive places in the environment and the edges represent direct paths between them. A significant issue in building topological maps is defining distinctive places. For example, if the robot traverses two places that look alike from the point of view of the sensor, topological mapping

methods often have difficulty determining if these places are the same or not (particularly if these places have been reached via different paths).

On the other hand, a bare graph does not provide much information, and therefore additional information for localization and navigation is usually associated with both nodes and edges. For example, a system capable of creating a topological map of a park-like environment [71] associates orientation information with nodes and path texture with edges. Other system, which associates visual information with nodes detected by sonar sensors is described in article [72].

The most significant advantage of topological maps is their scalability and simplicity. They allow to integrate large-scale area maps without suffering from the accumulated position error, and path planning can be easily resolved just using Dijkstra algorithm. Moreover, topological maps are suitable for fast, hierarchical planning and can be easily modified to include semantic information, but they are difficult to construct directly from sensor data and they are not suitable for precise localization and exact reconstruction of the environment.

2.3.3 Hybrid maps

There are numerous works combining metric and topological approaches into so called hybrid maps [73]. These approaches differ in the form the particular maps that are built, interconnected, and used. For example, in [74] the topological layer is built on top of a grid-based map by processing Voronoi diagrams of the thresholded grid. In [75], a statistical maximum likelihood approach is introduced, where a topological map solves global position alignment problems and it is consequently used for building a fine-grained map. The map representation in the SLAM framework Atlas [76] consists of a graph of multiple local maps of a limited size. Each vertex represents a local coordinate frame of reference and each edge denotes the transformation between local frames as a Gaussian random variable. For indoor navigation Youngblood [77] proposes a place-centric occupancy grid model, where each area or room of the environment is associated with a single occupancy grid. A global graph interconnecting these grids is used to represent the connectivity between rooms. The extent of a room is determined by retrieving a locally confined area followed by gateway extraction.

In [78] we present a novel mapping approach that is utilized for indoor exploration of a-priori unknown environment. This approach uses fixed-size interconnected occupancy grids, instead of associating one grid to each room. Since this grid decomposition does not intend to represent the topology of the environment, a separate graph is used for this end. In contrast to Youngblood's approach, our method does not rely on assumptions of the structure of walls

and enclosed areas for gateway and topology extraction. Furthermore, with a fixed-size grid approach, large rooms will not generate single large grids, negatively impacting on path-finding exploration.

2.4 Simultaneous Localization and Mapping

Localization and mapping are seldom addressed separately, in most cases the robot incrementally adds new information to a map and uses map-based localization to estimate its position. This is called Simultaneous Localization and Mapping (SLAM) [35] [36]. In SLAM framework the map is represented by the positions of the landmarks in the environment. As the localization error influences the quality of the map and vice-versa, the key issue of SLAM is dealing with uncertainty of the robot position and the map. Methods described in [79] use probabilistic methods to update robot knowledge about its position and the surrounding environment. Two major approaches to probabilistic modeling have been used: Extended Kalman (EKF) and Particle Filtering. Both approaches work sequentially, the probability distributions are updated in so-called prediction and measurement steps, which correspond to robot movement and sensor measurements respectively. During a prediction step, which corresponds to robot movement, the uncertainty in robot position increases whereas that measurement step typically causes the uncertainty to decrease.

The Extended Kalman Filtering (EKF) SLAM models both robot and landmark position by means of multidimensional Gaussian distributions. The world information is kept in a vector containing robot and landmarks positions and a covariance matrix of this state vector. Each time the robot moves or takes a measurement, the state vector and covariance matrix are modified. The computational cost increases with the map size. This is particularly painful in the case of naive SLAM implementations, where computation of the Kalman gain during map update requires inverting the covariance matrix. A huge variety of EKF based SLAM methods have been implemented, some focusing on speed [80, 81], some on precision [82] and other aspects [83]. The SLAM methods which do not solve loop closing are subject to drift and therefore are qualitatively comparable to odometry in terms of global position estimation.

The main disadvantage of Kalman filtering is that it can model only unimodal probability distributions. This limitation can be overcome by using Particle Filter. The idea is to keep several 'particles', each with a unique hypothesis of robot position and map [84]. Density of these particles correspond to probability distribution of the robot position. Each time the robot moves, the particles are moved and randomly distorted according to the robot movement model (this corresponds to the prediction step). In the measurement step, the sensory inputs are simulated for each particle and compared

to the real measurements. Based on this comparison, the particles can be discarded, duplicated or left alone. A major disadvantage of particle filtering is its computational complexity, because for proper uncertainty modeling a lot of particles have to be maintained.

The computational power of today's computers allows real-time image processing. Using image features as natural landmarks allows the development of a new technique of SLAM: visual SLAM. Some visual based methods use stereo cameras in order to obtain instant range information [85]. Other methods substitute stereo vision by motion and use a single (monocular) camera [44, 86]. However, most monocular approaches are computationally expensive and achieve low operational speeds when mapping large scale environments. This problem can be solved by dividing the large global map into smaller maps with mutual position information [87, 88].

In recent years, an alternative to sequential, frame-by-frame SLAM has been developed. Instead of rigidly linking feature tracking and mapping, these tasks are split into two separate threads which run asynchronously, leading to PTAM (Parallel Tracking and Mapping) approach. Tracking can focus on robustness and real-time performance. Mapping is done using key-frames and standard Structure from Motion (SFM) techniques (i.e. bundle adjustment) and thus scales very differently to standard EKF Visual SLAM [89].

2.5 Motion planning

Motion planning addresses the fundamental problem in robotics of translating high-level tasks in terms of human specifications into low level commands for robot actuators. A classical version of motion planning is sometimes referred to as the *Piano Mover's Problem*. Imagine giving a precise model of a house and a piano, the algorithm must determine how to move the piano from one room to another in the house without hitting anything.

In mobile robotics a basic motion planning problem is to produce a continuous movement that connects a starting configuration and a final configuration, while avoiding collision with known obstacles in the environment. A configuration describes a pose of the robot, and the configuration space is the set of all possible configurations. The robot and obstacle geometries can be described in a 2D or 3D workspace, while the motion is represented as a path in configuration space. The set of configurations that avoids collision with obstacles is called the free space. [90].

The problem of motion planning is usually decomposed into two subproblems, path planning and motion control.

2.5.1 Path planning

Path planning solves the problem of finding a traversable path from the current robot pose to the desired destination across the free space. Typically, the resulting plan is a sequence of points, which the robot has to move through in order to reach the goal. To plan a path, the algorithms need a map of the robot operational space. Exact path planning for high-dimensional systems under complex constraints is computationally intractable. However, there are some approaches that have been successfully tested under specific hypothesis among which we can mention grid-based, geometric, potential-field, sampling-based, graph-based, among others.

Grid-based approaches overlay a grid over the configuration space, and assume that each configuration is identified with a grid point. At each grid point, the robot is allowed to move to adjacent grid points as long as the line between them is completely contained within free space. This discretizes the set of actions, and then a search algorithm (like A*) can be used to find a path from the start to the goal. The number of points on the grid grows exponentially with the configuration space dimension, which makes it inappropriate for high-dimensional problems [34]. A geometric approach consists of thinking the path-planning as a problem of moving a polygonal object across polygonal obstacles in two or three dimensional [91] maps. Path planning for topological maps is based on classical graph searching methods like A*. The Potential Fields approach treats the robot configuration as a point in a potential field that combines attraction to the goal, and repulsion from obstacles. The resulting trajectory is output as the path. This approach has the advantage that the trajectory is produced with little computation. However, it can become trapped in local minima of the potential field, and fail to find a path [92]. The conventional potential field method is not suitable for path planning for a robot in a dynamic environment where both the target destination and the obstacles are moving, but there are some works that address this problem [93].

A major problem of the aforementioned decomposition lies on the fact that standard path planning does not take into account robot kinodynamic constraints and therefore might not generate time-optimal paths. For some cases of robot dynamic constraints, the generated path might not be traversable for the robot at all. This disadvantage is addressed by application of Rapidly exploring Random Trees (RRT) [94]. These algorithms are able to make plans for robots with many degrees of freedom, while respecting their kinodynamic constraints. However, this class of algorithms is slow unless a suitable heuristic is applied [95].

2.5.2 Motion control

Motion control algorithms generate robot actuator inputs (e.g. wheel speeds, motor Pulse-Width-Modulation PWM, input current to leg actuators) to move the robot along the set of points given by path planning. These inputs are generated by a controller. There is a variety of types of controllers. To name a few: closed-loop controllers, like the well known Proportional-Integral-Derivative (PID) [96], predictive controllers [97], fuzzy logic [98], neural network controllers [99], among others. The structure and parameters of these controllers are strongly influenced by robot physical parameters. Optimal selection of these parameters can be done either manually or by the controller itself. In this Thesis we will use the Proportional-Integral-Derivative controller to set the desired velocity for each motor of the robot (see Chapter 3).

PID controller

A Proportional-Integral-Derivative (PID) controller is a generic loop feedback controller widely used in control systems. A PID controller calculates an error value as the difference between a measured process variable and a desired setpoint. The controller attempts to minimize the error by adjusting the process control inputs. The PID controller algorithm involves three separate constant parameters, and is accordingly sometimes called three-term control: the proportional, the integral and derivative values, denoted P, I, and D. Heuristically, these values can be interpreted in terms of time: P depends on the present error, I on the accumulation of past errors, and D is a prediction of future errors, based on current rate of change.

Tuning a PID control loop means setting its control parameters P, I and D to the optimum values for the desired control response. Stability (bounded oscillation) is a basic requirement, but the main goal is that the controller reaches the desired setpoint in the least amount of control loops. PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control (the parameters are constant and there ir no direct knowledge or model of the process). There are accordingly various methods for loop tuning, and more sophisticated techniques are the subject of patents. In this Thesis we will use the Ziegler-Nichols tuning method. In Chapter 3 the implementation of a PID controller for the ExaBot motors is detailed.

2.6 Exploration

If the goal of the navigation is to create a complete map of the surrounding environment without steering the robot manually, the problem can be defined as autonomous exploration. An exploration algorithm can be defined as an iterative procedure consisting of a selection of a new goal to explore and a navigation to this goal. Such an algorithm is terminated whenever the defined condition (mission objective) is fulfilled. Besides, the usage of resources (e.g. the exploration time, the length of the trajectory) is optimized. The exploration strategy determines the next target position in each exploration iteration (one exploration step) with respect to the current robot position, the current knowledge about the environment (i.e. current map), and a selected optimization criterion. Any exploration strategy has to be able to adapt to any unexpected situations during map acquisition. The greedy algorithms are often used as an exploration strategy. The robot always moves from its current location to the closest location that has not been visited (or observed) yet, until the environment is mapped [100]. The exploration algorithm depends on the type of the created map; as we see in Section 2.3 it is possible to find both metric and topological (or even hybrid) exploration approaches.

The central question of exploration is where to place the robot in order to obtain new information of the environment. The exploration method implemented by Yamauchi [101] proposes to place the robot in a 'frontier' - a boundary between mapped and unknown environment. Finding a frontier on an occupancy grid map is a matter of simple mathematical morphology operation. Typically, several frontiers exist in the given map, which allows a simple extension of this approach to multi-robot exploration [102]. Frontier based exploration is not limited to the case of robots with rangefinding sensors only, the frontiers can be detected by vision-based systems as well [103]. Exploration approaches can be also based on the Next Best View algorithm, which is widely used in three-dimensional mapping [104, 62]. Another approach, based on the notion of entropy is called information-gain [105] exploration.

2.7 Vision-based navigation

As discussed at the beginning of this Chapter, autonomous navigation can be described as the process of determining a suitable and safe path between a starting and a goal point for a robot travelling between them. Different sensors have been used to this purpose, which has led to a varied spectrum of solutions. Active sensors such as sonars [106], laser range finders [107], radars [108] and 3D cameras based on time of flight[109] and structured light [110] are used in autonomous navigation methods. These sensors are inherently suited for the task of obstacle detection and can be used easily because they directly

measure the distances from obstacles to the robot.

However, none of these sensors can perfectly accomplish the navigation task and they all have disadvantages. Sonar sensors suffer from specular reflections and poor angular resolution. Standard laser range finders are precise, but they only provide measurements in one plane. Three-dimensional laser rangefinders, as well as most radars, are not suitable for small robot applications because of size, weight and energy consumption. Most 3D cameras illuminate the perceived scene with infrared light and do not work outdoors due to the presence of sunlight. The GPS and other beacon systems rely on external infrastructure or structured environment. All active sensors transmit signals which might interfere with each other if multiple sensors or multiple robots are present in the same environment. Moreover, the distance measurements provided by these sensors are not suitable to distinguish between different types of ground surfaces or recognize the shape of a road without the presence of bounding structures such as surrounding walls. The shaft encoders, gyros and compass sensors are suitable, but cannot be used over long paths or long periods of time because their measurement error grows over time.

On the other hand, visual sensors are increasingly affordable, they are small and can provide higher resolution data and virtually unlimited measurement ranges. They are passive and therefore do not interfere with each other. Most importantly, visual sensors can not only detect obstacles, but also identify forbidden areas or navigate mobile robots with respect to human-defined rules (i.e. keep off the grass). Such forbidden areas are not obstacles, since they are in the same plane as the path, but should be considered as non-traversable. For these reasons vision-based navigation has long been a fundamental goal in the field of mobile robotics research in last years. Vision-based navigation concept is closely related to the visual servoing, that can be defined as the use of the vision sensor in feedback control [111, 112].

As for other sensors, the different visual navigation strategies proposed in the literature make use of several configurations to get the required environmental information to navigate. Most systems are based on monocular and binocular (stereo) cameras, although systems based on trinocular configurations also exist. Another possible structure is omnidirectional cameras that are usually obtained combining a conventional camera with a convex conic, spherical, parabolic or hyperbolic mirror.

Traditionally, vision-based navigation solutions have mostly been devised for Autonomous Ground Vehicles (AGV), but, recently, visual navigation is gaining more and more popularity among researchers developing Unmanned Aerial Vehicles (UAV) and also Autonomous Underwater Vehicles (AUV). Regardless of the type of vehicle, systems that use vision for navigation can be classified according to different criteria. Referring to mapping, those that need previous knowledge or map of the environment, those that build a representa-

tion of the environment as they navigate through it and those that do not need a map at all. Also, there are systems that localize the robot in the environment and others that do not. Moreover, navigation solutions can be divided according to the environment: outdoor, indoor, structured, unstructured [113].

2.7.1 Map-based visual navigation

For map-based visual navigation it is usual to divide the problem into two phases, what is known as map-and-replay technique. With this approach the robot first traverses the desired path guided by a human operator in a training step. During this phase, the robot records visual landmarks so it can build a representation of the environment, i.e. a map. In the second phase, the robot can use this map to navigate autonomously through the learned path. There are lots of works that use this technique to achieve autonomous navigation. Most of them use local image features as visual landmarks [49, 114]. In Chapter 5 a map-and-replay method is presented in detail and in Chapter 6 the performance of local image features for long-term visual navigation is comprehensively evaluated.

There are other map-based visual navigation systems that incrementally build a map and simultaneously use this map to navigate in the environment. In map-building basic approach, it is assumed that the localization in the environment can be computed by some other techniques. When this assumption is removed, the visual navigation includes the exploration and mapping of an unknown environment. In this case the robot must accomplish three tasks: safe exploration/navigation, mapping and localization in a simultaneous way. We have already discussed this approach in Section 2.4 as visual SLAM. However, the main goal of SLAM is to simultaneously perform localization and mapping, and therefore, the robot is generally guided through the environment by a human-operator, which is actually not autonomous navigation.

2.7.2 Mapless visual navigation

Mapless visual-based navigation systems mostly include reactive techniques that use visual clues derived from the segmentation of an image, optical flow, or the tracking of image features among frames. Reactive systems usually do not need any previous knowledge of the environment but make navigation decisions as they perceive it. Those strategies process image frames as they gather them, and are able to produce enough information about the unknown and just perceived environment to navigate through it safely. Optical flow can be defined as the apparent motion of features in a sequence of images. During navigation, the robot movement is perceived as a relative motion of the field of view, and, in consequence, it gives the impression that static objects and

features move with respect to the robot. To extract optical flow from a video stream, the direction and magnitude of translational or rotational scene feature movement must be computed for every pair of consecutive frames [115]. The image segmentation approach uses a combination of color and texture pixel classification to perform a segmentation of the image in navigable and non-navigable zones. This technique is commonly used in outdoor navigation to follow the path [116, 117]. In Chapter 4 we present a novel segmentation-based visual navigation method for path following in outdoor environments. Finally, image navigation based on feature tracking uses features to track the movement of the camera (and of course of the robot) through the environment. This approach estimates the pose of the robot during navigation so it is closely related to visual odometry, as we already referred in Section 2.2.

2.8 Conclusions

In this Chapter we present the state of the art in Mobile Robotics. This extensive review allows us to categorize the methods associated with the main problems involved in mobile robotics. Mapping, Localization and Motion Planning issues are addressed and advantages and disadvantages of different approaches are described. This Chapter also shows that although most common solutions for autonomous navigation use many expensive sensors, it is also possible to achieve it using only standard digital cameras. Chapters 4, 5 and 7 of this Thesis will focus on monocular visual navigation.

Chapter 3

ExaBot: a new mobile robot

In this Chapter we present the ExaBot, its body, sensors, actuators and processing units. We also address the motion control of the robot, which is taken into consideration in the next Chapters where the ExaBot is used as an experimental platform.

3.1 Introduction

Mobile robots are commonly used for research and education. Taking as a premise that *"There is no robotics without robots"*, it is necessary to have platforms where different proposed methods can be tested. Nowadays, there are many commercial mobile robots available. However, these robots do not always meet the characteristics needed for certain tasks and are very difficult to adapt because they have proprietary software and hardware.

For example, Khepera [118] is a mini (around 5.5 cm) differential wheeled mobile robot that is developed and marketed by K-Team Corporation. The basic robot comes equipped with two drive motors and eight infrared sensors that can be used for sensing distance to obstacles, and also sensing light intensities. It is very popular and widely used by over 500 universities for research and education. However, Khepera robot serves only for indoor small environments and although some extensions can be added, it is very limited when modifications to its sensing or programming capabilities are needed.

Another example of a well-known commercial mobile robot is the Pioneer 2-DX and its successor Pioneer 3-DX [119]. They are popular platforms for education, exhibitions, prototyping and research projects. These robots are quite bigger than the Khepera (more than 10 times) and have a computer integrated into a single Pentium-based EBX board running Linux. This processor unit is used for high-level communications and control functions. For locomotion, the Pioneer robots have two wheels and a sonar ring as range

sensors. They can be used for both indoor and outdoor environments. Lots of accessories such as new sensors and actuators can be purchased from Adept MobileRobots manufacturer. Nevertheless, because of its size, Pioneer robots need a large workspace to move around and its weight makes it unsuitable to be transported around easily and tedious to be operated by a single human.

Besides the above disadvantages, the main drawback of these commercial mobile robots is their cost. For instance, a basic Pioneer robot costs approximately $\$5,000$ dollars, and a basic Khepera robot costs $\$4,000$. It is very difficult for latin american research labs and universities to afford these costs and hence this severely limits the possibilities of buying or upgrading these robots, for single and even more for multi-robot systems. Moreover, the maintenance of commercial robots can be very hard for developing countries. If some component of the robot breaks it is not easy to purchase the replacement, it could take a lot of time due to shipping, and this in turn may delay planned experiments. These issues are the main motivations for developing our own low-cost mobile robot in our Lab. Furthermore, the knowledge gained from the design and construction of a robot from scratch is also an important incentive for this task. The new platform should be affordable, then the relationship between the cost of the robot, its size and functional capabilities should be taken into account.

(a) Khepera II (b) Pioneer 3DX

Figure 3.1: Commercial and very popular mobile robots for research and education: (a) Khepera is a mini robot around 5.5 cm, (b) Pioneer is more than 10 times bigger.

Thus, given that commercially available robots do not have the characteristics needed for some tasks and are too expensive to acquire, repair or modify, the development of new robot prototypes for research and education becomes a relevant task. Additionally, the experience acquired by making a mobile robot was essential to get an autonomous navigation system, which is the main concern of this Thesis.

In this Chapter we present the ExaBot: a small mobile robot developed in the Robotics Lab at our University. The rest of the Chapter is organized

as follows: in Section 3.2 the design decisions are considered, in Section 3.3 the mechanical parts of the robot are detailed, in Section 3.4 we describe the general hardware architecture of the ExaBot and its possible configurations, in Section 3.5 we describe the different sensors available in the robot, in Section 3.6 we describe its actuators, in Section 3.7 motion control of the ExaBot is detailed, Section 3.8 shows some examples of the usage of the ExaBot in research, education and outreach activities and finally, Section 5.5 concludes this work.

3.2 Design Considerations

The goal of the ExaBot is to have one single robotic platform that allows to carry out research, education and outreach activities. To achieve this goal the robot should have a reconfigurable architecture that supports many different sensors and processing units, and can be easily reconfigured with a particular subset of them for a given task. On the other hand, the robot should be a low cost alternative compared to its commercial counterparts and have similar functionalities. Therefore, we address a three way design trade-off between size, cost and functionality.

3.2.1 Size

The body of the ExaBot should be small enough to be transported around easily, but also big enough to support many sensors and different processing units. On the other hand, the dimensions of the chassis should be large enough to allow an embedded computer inside. Also, the robot should be able to carry a laptop or a mini external PC on top of it. The relation of robot cost to robot size indicates that off-the-shelf components are the best option. If the platform is too small it gets expensive due to the advanced technologies and fabrication techniques required (e.g. microelectromechanical systems, micro assembly, etc.), and to the lack of off-the-shelf components. On the other hand, if the platform is too big, the cost of the chassis increases considerably and it also becomes more difficult to use because it requires a large workspace. Therefore, we decided for a small, not mini, size for the ExaBot. The final dimensions of the robot depend on the desired configuration, but it is around 25 cm (for more details see Section 3.3).

3.2.2 Cost

The cost is one of the main constraints of the robot design. It should be in the order of ten times less compared to its comercial counterparts. Thus, the

chassis, sensors and actuators of the robot should be inexpensive, but allow a wide application spectrum. Using a pre-built body, off-the-shelf components and developing the electronics of ExaBot on our own, it is possible to achieve this goal. Because the robot should be small we decided to use small, cheap and still readily available sensors. On the other hand, as the ExaBot has the ability to carry a laptop mounted on it, we can use it as the main processing unit, decreasing significantly the cost of the robot.

3.2.3 Functionality

When considering functionality one must consider the domain that the robot is expected to work in, and what it is expected to do in that domain. As we want a multipurpose robot, that can be used for a variety of activities, the ExaBot should be designed to support many different sensors and processing units, and to be easily reconfigured with a particular subset of them for a given task. Regarding locomotion, the robot should be able to operate in both indoor and outdoor environments. In order to fulfill autonomous navigation, it should be able to move forward, backwards and turn on the spot. However, there is no need for moving sideways. Thus, the simpler solution is to use the widely known differential drive with two wheels that cover the whole chassis. Moreover, since in some situations a harsher environment can be encountered, it could be desirable to get caterpillars instead of wheels. This locomotion allows the robot to go over small obstacles and traverse slippery floors, giving an advantage that can come in handy when working in outdoor unstructured environments. Regarding sensing, the ExaBot should have a variety of sensors including range sensors, contact sensors, vision sensors, linefollowing (floor) sensors, shaft wheel encoders, consumption and battery voltage sensors. Finally, the robot should have different processing capabilities depending on the task. Of course, microcontrollers are needed to handle the sensors and motors of the robot. Also, an embedded computer can be a good solution for high-level communications and control functions. The possibility of mounting a laptop over the robot makes a good choice for testing new navigation methods and research activities. In this case the embedded computer can be removed, reducing the cost of the robot.

In summary, considering cost, size, and functionality, we designed ExaBot as a small, low-cost, multipurpose mobile robot taking advantage of off-the-shelf components, developing the electronics on our own and keeping a good functionality level, comparable to its commercial counterparts, to a tenth of their prices.

3.3 Mechanical design

The body of the ExaBot should be rugged enough to be handled not only by researchers but also by inexperienced students, and fit the design considerations mentioned above. Since mechanical issues are not a goal of this work, the simplest solution that meets the constraints is preferred. Hence, a pre-built mechanical chassis was selected. We reviewed several models and decided to use the Traxster Kit [120]. This aluminium alloy chassis is light (900 grs) and small sized (229 mm length $\times$ 203 mm wide $\times$ 76 mm height), so it is transportable and can accommodate multiple sensors and processing units. It has two caterpillars, each connected to DC (Direct Current) motors (7.2V and 2A) with built-in quadrature encoders. The shape of the chassis results attractive to mount several sensors and carry a laptop or other mini PC on the top, an embedded computer inside, and the battery on the bottom. Of course, the election of this mechanical kit constrained future decisions, such as the layout and design of the control board and the type of embedded computer that can be mounted inside the chassis. More details about the motors and encoders of the kit can be found in Section 3.6. Last but not least, this kit costs around $200 dollars, which meets the low-cost requirements and it results cheaper than buying the chassis, motors and encoders separately.

Figure 3.2: The Traxster mechanical kit.

3.4 Hardware design

In this section we describe the hardware architecture of the ExaBot, the different processing units, communication buses and power supply. We also present two possible hardware configurations for the ExaBot.

3.4.1 Configuration with an embedded computer

The standard configuration of the ExaBot has an embedded PC104 computer as central processing unit and 3 Microchip PIC microcontrollers: one to control most of the sensors (PIC18F4680), and one to control each DC motor included in the Traxster Kit (PIC18F2431). Figure 3.3 shows this configuration.

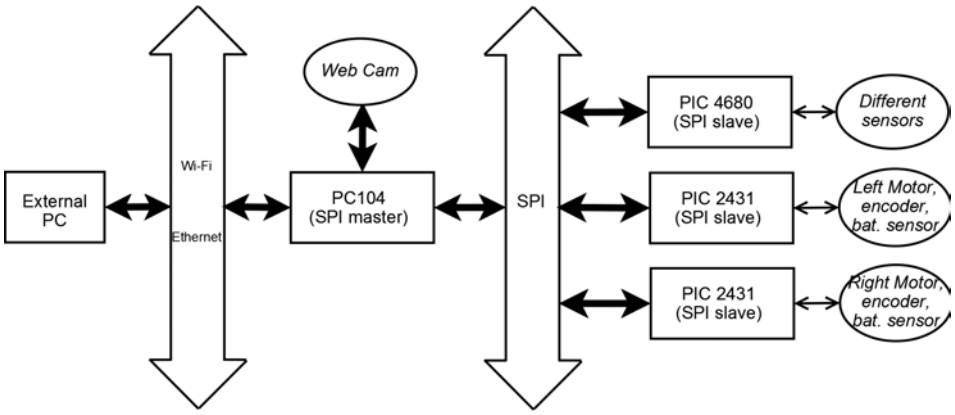


Figure 3.3: ExaBot architecture

Processing units

The different goals of the ExaBot pose very different computational power needs. Many research activities such, as vision-based navigation algorithms, are usually computationally demanding, while most of the outreach experiences can be done with very simple programs. The processing power is divided into two levels: low level processing units for sensor and motor control, and a high level processing unit for communications and robot high level control functions. Low level processing is performed by one PIC18F4680 microcontroller [121] that controls the sensors and two PIC18F2431 [122] that control each motor and their corresponding encoders. Despite the fact that a single PIC18F2431 can control both motors, we decided to use two in order to be able to export expansion ports to control further motors that might be added to the robot in the future. The high level processing unit should ensure general purpose programming. Thus, the most suitable CPU-based element for this goal is an embedded processor. An embedded ARM 9 PC104 of 200 Mhz TS-7250 [123] was included. This embedded PC has 2 USB ports, a serial port, an Ethernet port and several General Purpose I/O. It runs Linux Kernel 2.26. This embedded PC consumes a maximum of 400mA, which represents a reasonable consumption for a processor with those characteristics. This main processing unit can be easily removed or replaced (see Section 3.4.2).

Communications

The communication between the PC104 and the microcontrollers is done by means of a SPI (Serial Peripheral Interface) bus. This is a full duplex, serial and synchronous communication bus between one master and many slaves. We chose it over other possible serial buses such as I^2C or RS232 since it is much simpler to implement and it meets exactly what we need: fast communication between one fixed master (the PC104) and many slaves (the PICs). To communicate with external PCs, the ExaBot has a serial port, Ethernet and a Wi-Fi connections provided by the PC104 and a Wi-Fi USB key. Ethernet and Wi-Fi connections use UDP (User Datagram Protocol). For robot control, the timing of sending commands and the return of sensors data is vital. Unlike UDP, TCP (Transmission Control Protocol) is a connection-oriented protocol. If the received data is corrupt, TCP protocol requests the sender to resend corrupted data, causing delays in the communication. Receiving commands or sensor data late, as might happen with TCP, is far worse than eventually losing some command which might happen with UDP. For this reason we found UDP much more suitable than TCP for robot connection protocol.

Programming the robot

Programming the robot with this configuration is very simple since the PC104 has a Linux operating system. There are several open source cross-compilers and tool-chains to program embedded ARMs like the TS7250, for example using C++ and the `gcc` compiler.

3.4.2 Configuration without embedded computer

We include the embedded PC mainly to have enough processing power for some simple tasks. However, when more processing power is needed, the embedded PC can be removed and another external computer can be used as the main processing unit. In this case, high level control and communication algorithms are executed in the external PC. By removing the embedded PC, the robot cost is significantly reduced (the PC104 is about 20 percent of the cost of the robot) and the power consumption also decreases, making it possible for the ExaBot to run autonomously for longer periods of time. Two alternatives have been tested for this configuration: one using a laptop or netbook as a main processing unit and other mounting a Mini-ITX Asus Motherboard with GPU (Graphics Processing Unit) over the chassis of the ExaBot. In these cases the vision sensor (i.e. camera or laser) can be connected to the external PC. The architecture configuration without the embedded computer is shown on Figure 3.4.

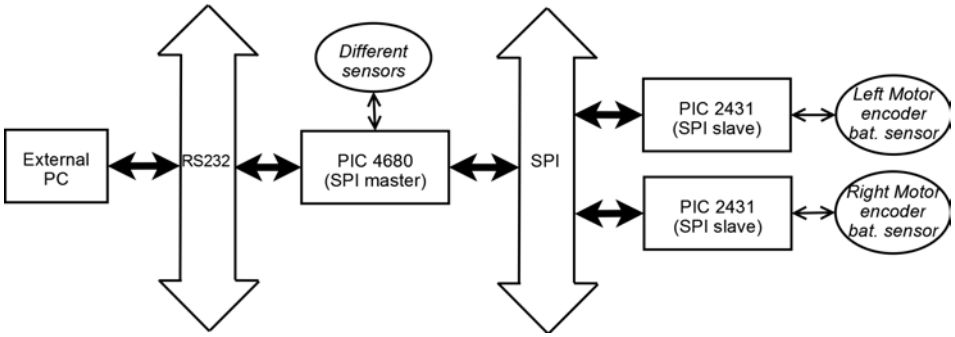


Figure 3.4: ExaBot architecture without the PC104

In this configuration, the robot's control is executed in the PIC that controls the sensors (PIC18F4680).

Communications

The communication between the PICs is done through the same SPI bus, but in this case the microcontroller of the sensors PIC18F4680 is programmed to be the master of the SPI bus. Communication with external computers is different. When the embedded computer is removed, a RS232 driver connected to the PIC18F4680 is used to provide an RS232 bus connection with external PCs. In this way, sensor data can be sent to the external computer where the high level algorithms can be run and commands to the ExaBot are sent again through RS232 (serial) bus.

Programming

In order to program the PIC18F4680 to execute the master of the SPI bus, we exported its programming interface, as well as the programming interface of the other PICs. There are several cross-compilers, tool-chains and IDEs to program these type of microcontrollers. This may prove very useful not only to run different tasks but also if changes to the base control of the sensors or motors are needed, or if new sensors are added to the robot (see Section 3.5.3). As the high-level algorithms are executed in the external PC, robot programming is made easier, particularly when the laptop is used as the main processing unit, in which case, re-compiling the code is not required.

3.4.3 Power

To supply power to the whole robot, rechargeable Ion-Lithium or Lithium-Polymer batteries can be used. At the beginning of the project, we used two

Ion-Lithium batteries. One 3 serial cells 1900mAh battery to deliver the 7.2V needed by motors and other 2 serial cells 1900mAh battery to deliver 5V to power all the electronic system, the embedded computer, the microcontrollers and sensors. At present time we are using more modern Lithium-Polymer batteries. In addition, if the Mini-ITX Asus Motherboard is mounted as an external computer, another battery is needed, in this case a 6 serial cells 2500mAh Lithium-Polymer 22.2V battery is used.

3.5 Sensors

A requirement of the ExaBot is that it should be able to be used for a wide variety of applications. Hence, we include different types of sensors in its design. Sensors can either be propioceptive (measure values internal to the robot) or exteroperceptive (acquire information from the robot's environment). The ExaBot has bumpers, white/black linefollowing (floor) sensors (i.e. linefollowings), rangefinders, a sonar and a vision sensor as exteroperceptive sensors and shaft enconders, current consumption and battery sensors as propioceptive sensors. It also has an external port that may be used to connect further sensors. All of these sensors can be dismounted and rearranged, turned off or even taken out of the robot if needed for particular tasks. In this way, we get a platform with the capability to use a wide variety of sensors, and at the same time that may use only a few required for a particular task.

As already stated, a PIC18F4680 microcontroller is used to handle the sonar, floor sensors, bumpers and the ring of 8 rangefinders. The encoders and motor current consumption sensors are controlled by a PIC18F2431. The vision sensor is directly controlled from the embedded PC104 or from the external PC. In the next Sections we will describe the sensors available in the ExaBot and hardware related issues.

3.5.1 Exteroceptive sensors

The range sensors chosen for the ExaBot are infrared rangefinders and a sonar. The rangefinders are short range point sensors (the range varies depending of the model, but is less than a meter). Sonars are non-point, long range (several meters) sensors. Both rangefinder and sonars are cheap sensors. Also, rangefinders have faster response than sonars. We chose to install a sonar in the front of the robot in order to achieve a long range sensing capability in the front, and a ring of 8 rangefinders to sense the environment surrounding the robot in a short range. In this manner, the robot has two types of range sensors and takes advantage of both.

Ring of rangefinders

The chosen rangefinders are the Sharp GP2D120 [124] with a sensing range of 4 to 30 cm. The rangefinder returns a voltage value proportional to the distance of the nearest object. This is a non-linear, analog function. Analog output voltage vs. distance to reflective object can be seen in Figure 3.5. As this is not a injective function, the rangefinders are placed 4 cm from the border of the chassis to disambiguate the values. In order to digitize the values, we use the A/D converter of the PIC18F4680. We also calibrated the rangefinders before mounting them on the robot and to this end we built look-up table that gives linear distance as a function of the digitized voltage. Each rangefinder gives a new value every 38 ± 9.6 ms, and has an unpredictable value between measurements.

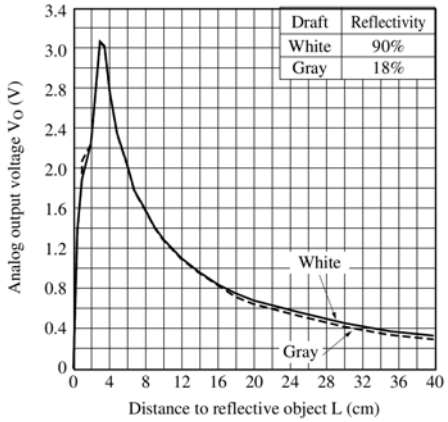


Figure 3.5: Analog output voltage vs distance to reflective object of the Sharp GP2D120, taken from [124].

Sonar

The chosen sonar is the Devantech SRF05 [125] that has a sensing range of 10 mm to 4 meters. There are sonars with larger sensing ranges (like SRF10) but are considerably more expensive, so they were discarded. The sonar works in the frequency of 40 KHz (wavelength 8.65 mm). It can be triggered every 50ms, giving a maximum of 20 measurements per second. The sonar output is a digital pulse which a duration that is linearly proportional to the distance to the closest object. Therefore, to control it we use a CCP (Capture/Compare/PWM) module, and a timer to measure the length of the sonar pulse, thus obtaining the distance to the closest object. The beam pattern of the sonar SRF08 is conical with the width of the beam being a function of the surface area of the transducers and is fixed. The beam pattern of the transducers used on the SRF08, taken from the manufacturers data sheet, is shown below.

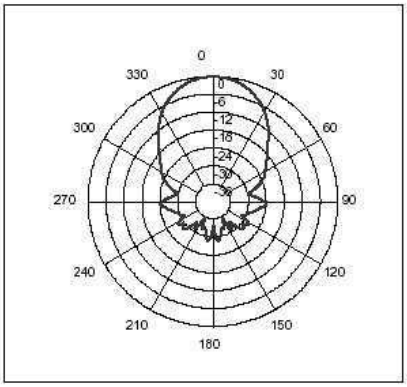


Figure 3.6: The beam pattern of the Devantech sonar SRF08, taken from [125].

Line following sensors and contact sensors

The ExaBot has two line following sensors and two contact sensors (bumpers) that are placed in the front of the robot. These sensors are connected to interrupt-on-change pins of the PIC18F4680. In this way, whenever a bumper is pressed or a line found on the floor, the pin changes its value and an interruption is generated.

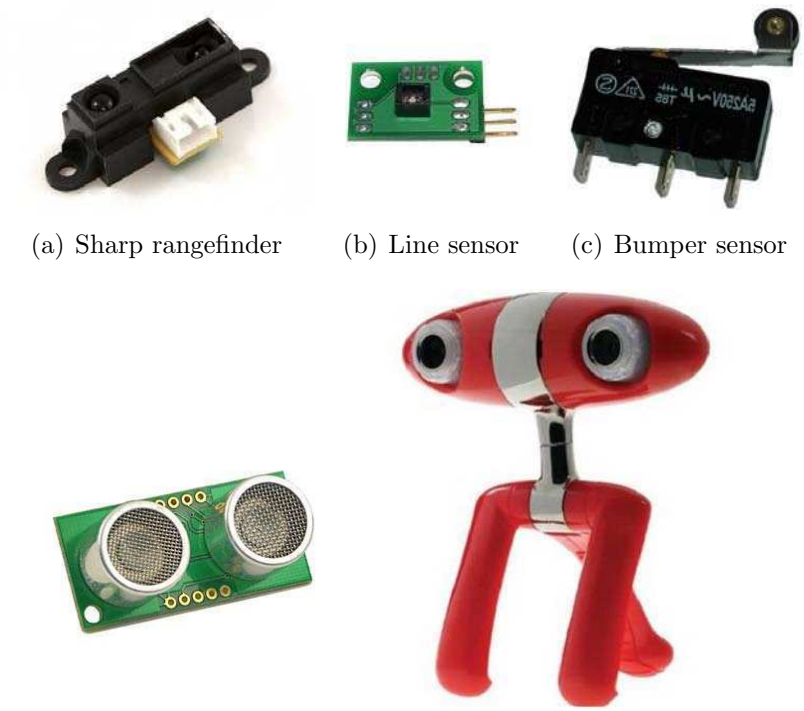
Vision sensor

The ExaBot has already been successfully used with three types of digital cameras: standard USB webcams connected directly to the embedded computer or external laptop, a stereo Minoru USB camera connected to the external PC and a firewire camera model 21F04, from Imaging Source, connected to a ExpressCard in the Mini-ITX motherboard (see Figure 3.3).

3.5.2 Propioceptive Sensors

Wheel shaft encoders

Wheel quadrature encoders (built-in in the Traxster Kit motor) where included to measure the movement of the motor. An encoder is an electro mechanical device that converts the angular position of a shaft to an analog or digital code. Incremental encoders provide information about the motion of the shaft, that is, they count the pulses received since the last time the sensor was checked. Quadrature encoders also provide information about the direction of the movement. In the case of the Traxster Kit, it included quadrature encoders that have a resolution of 624 Pulses per output shaft revolution. Using a 63.5mm diameter wheel (standard Traxter Kit wheels), it can achieve



(a) Sharp rangefinder (b) Line sensor (c) Bumper sensor

(d) Sonar SRF05 (e) Minoru stereo webcam

Figure 3.7: ExaBot exteroceptive sensors.

around 0.32mm of linear displacement per pulse of quadrature encoder, which represents a nice resolution for odometry pose estimation.

Battery sensors

The ExaBot has two battery sensors, one for the motor power and the other for the electronics and processing units. In this manner, it can be detected when the battery is running too low to continue and an alert can be issued by lighting a LED.

Motor consumption sensor

We implemented current sensor for the motors in order to have a control of the actual consumption and in that way implement a load control, and also a fault circuit. To sense the current we use an Allegro ACS712 current consumption sensor [126]. This sensor is connected between the L298 driver and the motor, it senses the current and outputs a voltage indicating the current. This voltage is used in two ways. For one, it is an analogical input to the PIC, that is digitalized and used to know how much current is consumed, and secondly it is used to implement a fault circuit in order to override PWM output and

hence stop the motors if error conditions happen (see Section 3.7).

3.5.3 Sensor expansion ports

To control the rangefinders, sonar, line-following, bumpers, and all the needed electronics to make the PIC18F4680 a possible master of the SPI bus (SPI chip select pins plus the control of the RS232 bus), all the 40 pins of that PIC are used. However, the two PICs that control the motors have several pins that are not used. We planned the pins needed for motor control in those PICs in order to maximize the possible applications of exported pins.

In this way, the ExaBot has two expansion ports, each one exporting the following:

- one analog pin: to connect any analog sensor like another rangefinder, a gyroscope, light sensors, etc.
- one CCP (Capture/Compare/PWM) pin module like the one used to control the sonar, to connect a another one, or a digital compass, for example.
- a PWM (Pulse Width Modulation) pair module, like the ones used to control the motors.
- the INT0 pin: an external interruption pin, in order to program an external interruption.
- GND and Vcc to power up further sensors.

These expansion ports may prove a very useful tool to add sensors and functionality to the ExaBot. The programming needed to control these added sensors may be done with the exported programming ports for each PIC18F2431.

3.6 Actuators

ExaBot actuators consist of two DC (Direct Current) motors that pull the caterpillars of the Traxster kit. These are 7.2V motors that draw a maximun current of 2A (300mA without load). They feature a 7.2Kg/cm torque, at 160 rpm without load, and come with a built in quadrature encoder of 624 pulses per output shaft revolution. A PIC18F2431 microcontroller connected with a L298 driver [127] is used to control each motor and corresponding quadrature encoder. The driver is needed because a PIC can typically drive up to 15mA at 5V and the motors consume up to 2A at 7.2V. Moreover, the driver has a H-bridge that allows the motor to run forward or backwards. Figure 3.8 shows the DC motor of the robot.

Figure 3.8: Exabot actuator: 7.2v DC Gearhead Motor

3.7 Motion control

Motion control should generate motor inputs. In the ExaBot we use Pulse Widht Modulation (PWM) and Proportional Integrative Derivative (PID) to control the actuators of the robot.

3.7.1 Pulse Width Modulation

In order to set the desired velocity for each motor, PWM technique can be used. Instead of generating an analog output signal with a voltage proportional to the desired motor velocity, it is sufficient to generate digital pulses at the full system voltage level (in this case 7.2V). These pulses are generated at a fixed frequency. By varying the pulse width, the equivalent or effective analog motor signal is changed and therefore the motor speed is controlled. The term duty cycle describes the proportion of 'on' time to the period of PWM; a low duty cycle corresponds to low speed, because the power is off for most of the time. Thus, the motor behaves like an integrator of the digital input impulses over a certain period.

3.7.2 PID controller

The ExaBot uses a PID loop feedback controller to reach the desired velocity for each motor (desired state). To do that the duty cycle for the PWM module has to be defined (control signal). To know the actual motor velocity the quadrature encoder data is read (feedback signal) from the QEI module. Thus, the input of the PID controller is the quadrature encoder data (actual velocity)

and the output is the next PWM duty cycle (desired velocity). The PID controller calculates an error value as the difference between a quadrature encoder data and a desired velocity. The controller attempts to minimize the error by adjusting the process control inputs. The proportional, integral, and derivative terms are summed to calculate the output of the PID controller. Defining $u(t)$ as the controller signal or output, the formal equation of the PID algorithm is:

$$u(t) = u(t-1) + K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt} e(t) \tag{3.1}$$

where K_p is the proportional gain, K_i is the integral gain, K_d is the derivative gain, e is the error (desired velocity - actual velocity), t is the instantaneous time and τ is the variable of integration that takes values from 0 to the present t. Each of the three terms of the equation can be interpreted as follows: the first term depends on the present error, the term second depends on the accumulation of past errors, and third term is a prediction of future errors, based on current rate of change. For completeness, Figure 3.9 shows the general schema of the PID controller.

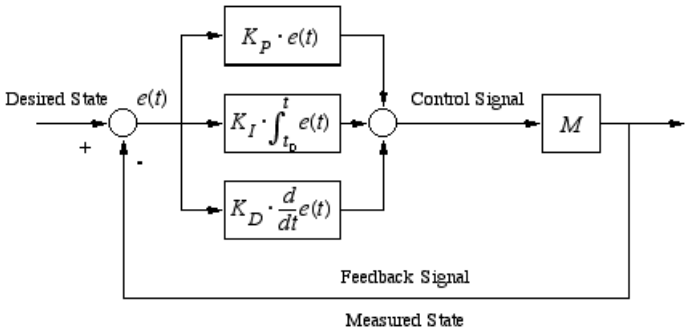


Figure 3.9: General schema of the PID controller.

The analysis for designing a digital implementation of a PID controller in a microcontroller requires the standard form of the PID controller to be discretized. Approximations for first-order derivatives are made by backward finite differences. The integral term is discretized, with a sampling time Δt, as follows:

$$\int_0^t e(\tau)d\tau = \sum_{\tau=1}^{t} e(\tau)\Delta t \tag{3.2}$$

and the derivative term is approximated as:

$$\frac{de(t)}{dt} = \frac{e(t) - e(t-1)}{\Delta t} \tag{3.3}$$

Thus, if the PID is computed for each control loop, sampling time $\Delta t = 1$ and we can replace in equation (3.1) to obtain:

$$u(t) = u(t-1) + K_p e(t) + K_i \sum_{\tau=1}^{t} e(\tau) + K_d(e(t) - e(t-1)) \tag{3.4}$$

Using this discretized equation a pseudocode for the PID controller is presented.

Algorithm 1 Simple PID controller pseudocode

```
previous_output ← 0
previous_error ← 0
integral ← 0
loop
    error ← desired_velocity - measured_velocity
    integral ← integral + error
    derivative ← error - previous_error
    output ← previous_output + K_p . error + K_i . integral + K_d . derivative
    previous_error ← error
    previous_output ← output
end loop
```

Then, the main problem is to set the K_p (proportional), K_i (integration) and k_d (derivative) coefficients. PID tuning is a difficult problem, even though there are only three parameters and in principle is simple to describe, because it must satisfy complex criteria within the limitations of PID control: the parameters are constant and there is no direct knowledge or model of the process. There are accordingly various methods for PID tuning, and more sophisticated

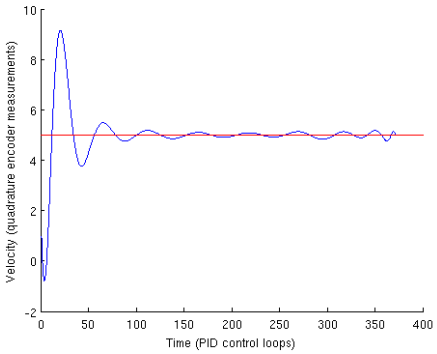
techniques are the subject of papers and patents. For this work we use the well-know Ziegler-Nichols tuning method, which is a heuristic developed by John G. Ziegler and Nathaniel B. Nichols. It is performed by setting the K_i and K_d coefficients to zero. Then K_p is increased (from zero) until it reaches the ultimate gain K_u, at which the output of the control loop oscillates with a constant amplitude and a period P_u. Then, the oscillation period P_u and the ultimate gain K_u are used to set the K_p, K_i and k_d gains following:

$$K_p = \frac{3}{5}K_u$$
$$K_i = \frac{2K_p}{P_u}$$
$$K_d = \frac{K_p P_u}{8}$$

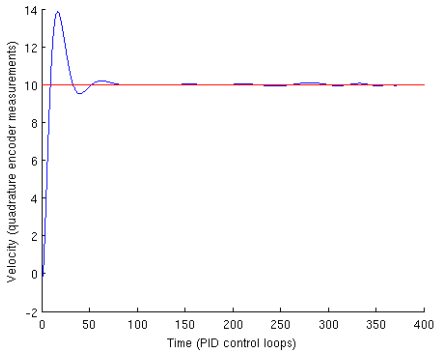
Using this method we experimentally found $K_p = 8$, $K_i = 4$ and $K_d = 400$. The next step is to test the obtained PID controller for different velocities in the ExaBot motors

3.7.3 Experimental results

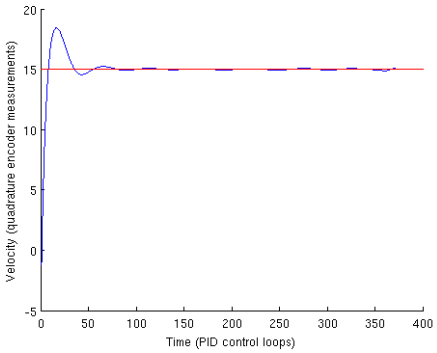
The PID controller has been tested for different velocities in experimental trials. Velocities are measured in quadrature encoder pulses, given by QEI module. For the ExaBot quadrature encoders resolution, the motor velocity can vary from 0 to 30 encoder pulses. In Figure 3.10 the results are shown. These experiments have been performed with the motors mounted in the robot, without load.



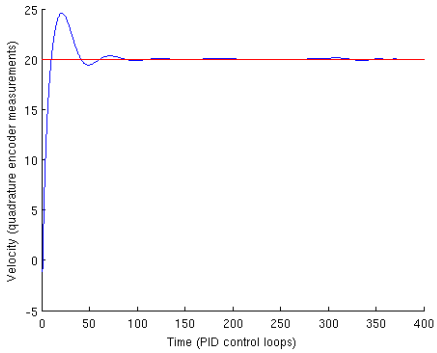
(a) Desired velocity = 5 pulses



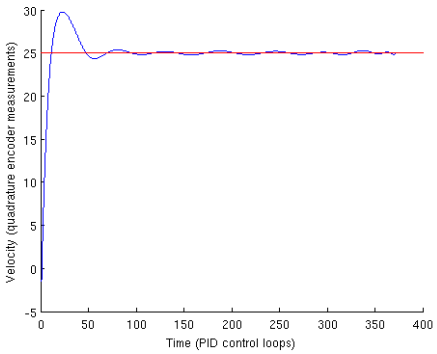
(b) Desired velocity = 10 pulses



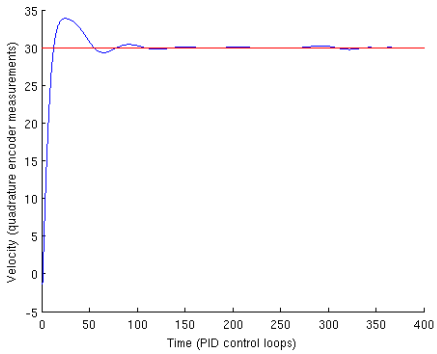
(c) Desired velocity = 15 pulses



(d) Desired velocity = 20 pulses



(e) Desired velocity = 25 pulses



(f) Desired velocity = 30 pulses

Figure 3.10: PID controller tests: Desired velocity (measured in quadrature encoder pulses) vs Time (measured in PID control loops)

As shown in the Figure 3.10, the PID controller fits the duty cycle for the PWM module to achieve the desired velocity. In less than 100 PID control loops the actual velocity converges to the desired one. The PID control loop is configured to be 10× the frequency of the PWM, that is a frequency of 0.1225 Khz (period of 8.1 ms). Then, as 100 PID control loops takes 810ms, our PID controller can ensure that the desired velocity is reached in less than one

second. Using this frequency for PID control, if we set the maximal velocity for the motors (30 encoder pulses) the robot reaches the maximal linear velocity of approximately 1,30m/sec. Contrary, if we set the minimal velocity for the motors (1 encoder pulse) the robot runs at the minimal linear velocity of approximately 0,043m/sec.

3.8 Aplications

To date six prototypes of ExaBot were built. Different configurations have been tested in several applications fulfilling all the goals the ExaBot was designed for (see Figure 3.11). These applications may be divided in three areas: Research, Education and Outreach.

3.8.1 Research

The main research activities with the ExaBot are related to autonomous visual navigation. In this Thesis we use the ExaBot as a experimental platform in trials for presented navigation methods. Both image segmentation based and image feature based algorithms have been successfully tested using the ExaBot. For some experiments, the robot was configured to use an external Mini-ITX board (AT5ION-T Deluxe) that includes a 16 core NVIDIA GPU as a main processing unit, and a firewire camera (model 21F04, from Imaging Source) as the only exteroperceptive sensor. The embedded PC104 computer was removed and the external AT5ION-T computer was connected using the serial port of the ExaBot board and RS232 protocol (see Figure 3.11). Another work presents the use of disparity and depth maps for autonomous exploration using a stereo camera [128]. In this work, a standard notebook or netbook is used as a main processing unit and a cheap Minoru 3D USB webcam as the only exteroperceptive sensor. The notebook connects through cabled ethernet to the PC104 (see Figure 3.11). Finally, monocular SLAM method is being implemented using as an Android-based smartphone as processing unit. In this case the camera of the smartphone is used as main sensor and the commands for the robot are given through a Wi-Fi connection (see Figure 3.11).

3.8.2 Education

The ExaBot is used in undergraduate and graduate courses of the Departamento de Computación, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. In particular, in a Robotics Vision course that covers several topics on computer vision applied to mobile robotics, the ExaBot has been successfully used in two semesters during 2011 and 2012 and will be used

this year again. Both monocular and stereo cameras have been used as main sensors for this course [129]. The ExaBot was also used in Artificial Neural Netwoks course where the range sensors were trained using multi-layer perceptron for obstacle avoidance. Finally, it has also been used in short lessons in other courses to show microcontroller programming guidelines.

3.8.3 Outreach

In the last years, robotic-centered courses and other outreach activities were designed and carried on. Three eight-week courses, five two-days courses, more than ten one-day workshops and talks were taught to different high school students using the ExaBots and ERBPI (Easy Robot Behaviour-Based Programming Interface) [130]. The ERBPI is a novel application for programming robots. Taking a behavior-based approach allows the user to define behaviors for the robot and encapsulates the low-level step-by-step programming. The application implements the idea of Braitenberg vehicles, where the robots sensors and actuators are simple abstract components that can be connected to each other to achieve a reactive behavior. Also, the ExaBot was part in several science exhibitions such as ExpoUBA, TEDxRíodelaPlata, Innovar and Tecnópolis. These activities are part of a comprehensive outreach program conducted by the Facultad de Ciencias Exactas y Naturales, UBA. As a result of this, hundreds of untrained people were able to program the ExaBot to have 'intelligent' behavior. This represented a comprehensive test that tried out the robustness of the ExaBot.

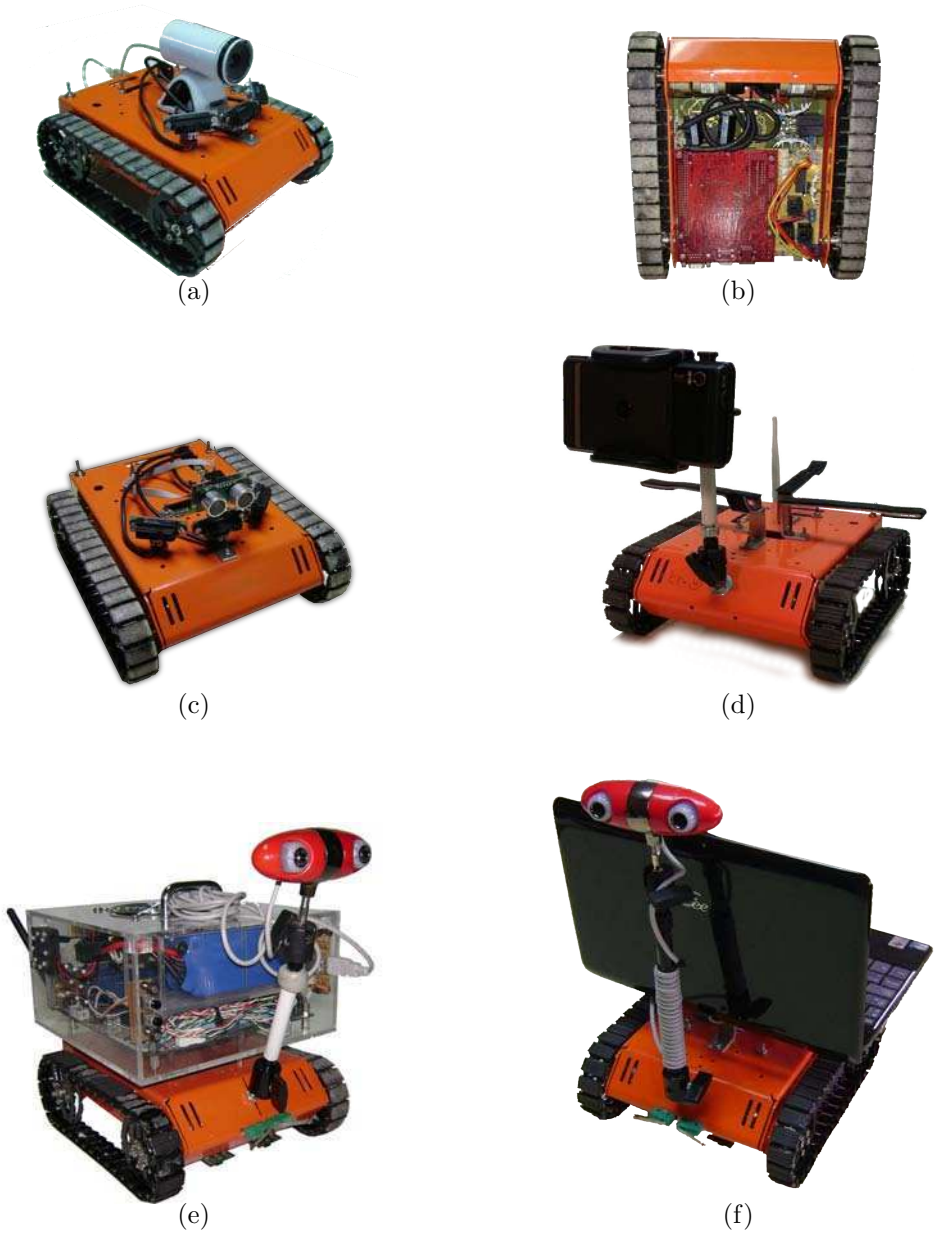


Figure 3.11: Different configurations of the ExaBot for different aplications: (a) ExaBot mounted with a webcam and three rangefinders, (b) Bottom view of Exabot without cover where embedded PC104 computer can be seen (red motherboard) (c) Exabot mounted with the sonar and three rangefinders, (d) Exabot with a smatphone as processing and sensing unit, (e) Exabot with an external Mini-ITX GPU board (AT5ION-T Deluxe) and a Minoru stereo camera, (f) mounted with a netbook as main processing unit and a Minoru stereo camera as a main sensor.

3.9 Conclusions and future work

In this Chapter we presented the ExaBot: a new mobile robot for research and education. The design of the ExaBot follows the premise that the robot must be usable to pursue very different tasks, has a comfortable size and low cost compared to commercially available robots with similar functionalities.

The hardware architecture is reconfigurable to include an embedded computer or remove it when it is not necessary, reducing the robot's cost and power consumption. It has a wide variety of sensors that can be reconfigured, even taken off, and it also has expansion ports that may be used to add further sensors. All the programming ports are exported. The robot may be programmed and controlled in three levels: from an external PC or laptop, from the embedded PC or from the PIC18F4680 microcontroller. These characteristics make the ExaBot a multipurpose robot, fulfilling its goal to serve for research, education and outreach activities.

The motion control of the ExaBot is based on a PID controller that controls the PWM duty cycle to determine each motor speed. For finding the proportional, integrative and derivative coefficients of the controller the well-known Ziegler-Nichols tuning method was used. The implemented algorithm of the PID controller was tested in the real robot for different velocities. The results show that the developed PID controller can ensure that the desired velocity is reach in less than one second.

The main characteristics of the new mobile robot can be summarized as follows:

- Dual-motor small chassis of $229 \times 203 \times 76$ mm (length, wide, height).
- Reconfigurable hardware architecture for a variety of purposes.
- Maximal linear velocity: 1,30m/sec. Quadrature encoder resolution: 0.32mm of linear travel per pulse of encoder.
- Sensing capabilities: ring of 8 rangefinders (range 5 cm to 80 cm), sonar (range 1 cm to 4 meters), 2 linefollowing sensors, 2 contact sensors, wheel shaft encoders, battery sensor.
- Lithium-Polymer Battery (1 - 2 hours)
- Embedded PC with a 200 Mhz ARM9 CPU, 32 MB SDRAM, 32 MB on-board flash drive, 2 USB ports, 2 serial ports, 1 Ethernet port and wi-fi. Operating System: Linux 2.6.
- Fully programmable for autonomous operation.
- Wireless remote control up to 100m indoors (line of sight)

Ongoing work on the ExaBot project includes adding further sensors such as a gyroscope, a digital compass and a indoor laser range finder; and the replacement of the embedded PC by a cheaper and more powerful model, like the RaspberryPi board.

Chapter 4

Segmentation-based visual navigation for outdoor environments

In this Chapter we present a novel autonomous navigation method for outdoor environments. This method allows a mobile robot equipped only with a monocular camera to follow structured or semi-structured paths. It does not require training phase, camera calibration or any *a priori* given knowledge or map of the environment to maximize its adaptability.

4.1 Introduction

As mentioned in Chapter 2, visual navigation problems are highly associated to the environment. Indoor robots may take advantage of architectural qualities to represent the environment highlights in terms of simple geometric features (i.e. lines, walls, floors) and use this guidelines to control the robot motion. Reversely, conditions imposed by completely unstructured outdoor environments (i.e. planetary exploration) involve more complex navigation strategies, with 3D terrain modeling, terrain classification, 3D path planning, stability problem, among others.

For completely unstructured environments and without imposing any assumptions about the workspace, a mapless image segmentation approach is not sufficient and fails. However, if we consider some assumptions, image segmentation can be a powerful tool to perform autonomous navigation. In this work we consider that the robot navigates in outdoor flat spaces that have been modified by the presence of humans. Such places are commonly characterized by a set of regions connected by a network of dirt track, tarred or tiled paths. For these types of environments, path identification and tracking along

Figure 4.1: ExaBot robot performing autonomous navigation in an unknown outdoor environment. The horizon line (red) is detected as well as contour of navigable space (blue). Middle points of the road are computed (yellow) to guide the robot. To achieve real-time constraints imposed by robot motion, the image segmentation is implemented on a low-power on-board GPU (on top of the robot).

an image sequence is a central issue. Image segmentation approach uses a combination of color or texture pixel classification to perform a segmentation of the image in navigable and non-navigable zones, hence it perfectly fits for the problem at hand.

The outline of the Chapter is as follows: in Section 4.2 the main related works are mentioned. Section 4.3 gives a brief overview of the proposed method. In Section 4.4 the horizon detection algorithm is described. Section 4.5 discusses different approaches for image segmentation, justifies the choice of graph-based segmentation for CPU and an optimized implementation of Quick shift algorithm for GPU. Section 4.6 deals with the problem of the classification of each super-pixel in traversable and non-traversable region. Section 4.7 explains how to extract the path contour from the classified image. Section 4.8 proposes a control motion to maintain the robot in the middle of the path. Section 5.5 shows experimental results with an image dataset of various outdoor paths as well as with the ExaBot mobile robot. Finally, Section 5.6 concludes the Chapter.

4.2 Related work

There are several previous works that propose autonomous vision-based navigation methods targeted towards outdoor environments using monocular cameras as their main sensor and without requiring any given knowledge or map of the environment. Most of them try to recognize the path's appearance as

well as its boundaries without any *a priori* information. Some basic methods rely on recognizing the edges that separate the road from its surroundings by identifying lines in the image (for example, using the Hough transform) [131]. But this approach is not valid for unstructured outdoor paths where the edges are not easily distinguishable. In some cases, the road is simply estimated as a geometrical shape as in [132] where an histogram is utilized for differentiating between the road region defined as a triangle and its two flanking areas. In addition, there is also a set of techniques that try to recognize the road by using the road's vanishing point [133, 134, 135]. In general these techniques rely on detecting converging edge directions to vote for the most likely vanishing point. The biggest problem of this type of approach appears when the road is not straight but curved or its edges are not well structured. Other systems detect the road by performing color and texture segmentation [117, 136]. Again, the downside of these methods is that they usually involve computationally expensive algorithms that could not be implemented on a mobile robot with limited computational equipment. Finally there are some works that merge both image segmentation and vanishing point detection. Using this approach, in [116] better results are reached and computation is performed on-board in a mobile robot, but the robustness of the system depends on the correct computation of the vanishing point.

Other works perform visual navigation using pixel classification. By assuming that the robot is operating on a flat surface, the problem can be reduced to classifying pixels into two classes: traversable or non-traversable. This approach, that is suitable for robots that operate on benign flat terrains, has been used in a variety of works for indoor navigation. In [137] classification is done at the pixel level. First, the image is preprocessed with a Gaussian filter. Second the RGB values are transformed into the HSV (hue, saturation, and value) color space. In the third step, an area in front of the mobile robot is used for reference and valid hue and intensity values inside this area are histogrammed. In the fourth step, all pixels of the input image are compared to the hue and intensity histograms. A pixel is classified as an obstacle if its hue or intensity histogram bin values are below some threshold. Otherwise the pixel is classified as belonging to the ground. This method is fast, as no complex computation is involved. The main drawback of this method is its low robustness to variable illumination and noise.

Due to these inconveniences, the idea of segmenting the image into a number of super-pixels (i.e., contiguous regions with fairly uniform color and texture) arises. In [103] a graph-based image segmentation algorithm [138] is used for indoor navigation in structured environments. Initially, an image is represented simply by an undirected graph, where each pixel has a corresponding vertex. The edge is constructed by connecting pairs of neighboring pixels. Each edge has a corresponding weight, which is a non-negative measure of the dissimilarity between neighboring pixels. Beginning from single-pixel regions,

this method merges two regions if the minimum intensity difference across the boundary is less than the maximum difference within the regions. Once the image is over-segmented in super-pixels, each one is labeled as belonging to the ground or non-ground region using the HSV histogram approach. While this method is more stable and robust, it is computationally quite expensive, so the exploration algorithm that uses this method has to stop the robot periodically to capture and process images of the workspace. Using the same image segmentation algorithm, in [139] super-pixels that are likely to be classified equally are grouped into constellations. Constellations can then be labeled as floor or non-floor with an estimator of confidence depending on whether all super-pixels in the constellation have the same label. This is a more robust method, but computationally expensive. Again, the robot must be stopped periodically to perform computations.

Finally, there is a group of works that perform visual-navigation using a probabilistic approach. In [140] a visual model of the nearby road is learnt using an area in front of the vehicle as a reference. A model is built which is subsequently used to score pixels outside of the reference area. The basic model of the road appearance is a mixture of Gaussians (MoG)-model in RGB space. This approach was used by Thrun in his Stanley robotic vehicle, which won the DARPA Grand Challenge in 2005.

Some of the aforementioned ideas from previous works are used to develop a novel method for real-time image-based autonomous robot navigation for unstructured outdoor roads. In outdoor unstructured environments, the navigable area is often cluttered by small objects with a color of the forbidden area, for example grass, tree leaves, water or snow in the middle of the path. In this case, most classification methods working at a pixel level would perform worse than methods which first segment the image into several areas with similar texture or color. Since such image segmentation is computationally expensive, we propose both CPU and also GPU-based embedded solutions to achieve the real-time constraints imposed by robot motion. To achieve a robust classification of the superpixels, a probabilistic approach similar to [140] is implemented, but using HSV color space instead of RGB. From all closed super-pixels classified as navegable path, a contour of this final region is extracted. Finally, middle points of this contour are found to compute the motion of the robot using a simple yet stable control law.

4.3 Method Overview

The proposed method processes an input image in order to obtain a robot control command as its final output. Figure 4.2 shows the image processing pipeline. Each step of the pipeline is briefly described here and in detail in the following sections:

Horizon detection. The input image is analyzed to find the horizon and then it is cropped such that only the region below the horizon is fed to the rest of the pipeline (see 4.2(b)).

Color space conversion & Smoothing. Along the RGB cropped image, an HSV version is also obtained. In the following steps, the HSV version is almost always used, unless otherwise specified. A median-blur filter is applied to the RGB image, in order to reduce noise, without smoothing edges. A blured HSV version is also obtained.

Segmentation. The RGB version of the blured image is segmented. This step produces a segment map, which consists of an image of labeled pixels (see 4.4(a)). The segment map is processed in order to build a list of segments or super-pixels, which describe groups of pixels with the same label. While building this list, mean and covariance (both in RGB and HSV color spaces) are computed for each segment.

Classification. A rectangular region of interest (ROI) corresponding to the area directly in front of the robot gives the system an example of the path appearance. The RGB/HSV information of this region is modeled by a mixture of gaussians (MOG). By comparing each super-pixel in the image to this MOG model, a classification criteria is applied to decide weather a super-pixel belongs to the path or non-path classes. The result of this classification is a binary mask (see 4.2(j)).

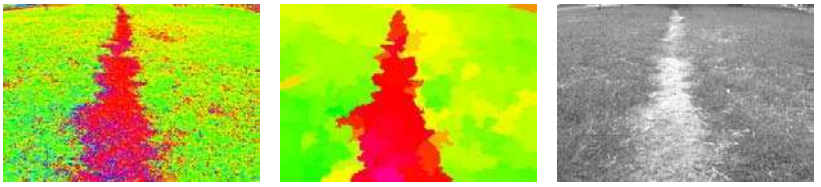
Contour extraction. From all closed regions classified as path, the most likely one is selected and a morphological opening filter is applied in order to 'close' small gaps. The contour of this final 'navigable path' region is found (see 4.2(k)).

Motion control. Middle points of this contour are extracted, which define a trajectory for the robot in the image. Finally, a simple yet stable control law sets the linear and angular speeds in order to guide the robot through the middle of the detected path (see 4.4(d)).

4.4 Horizon detection

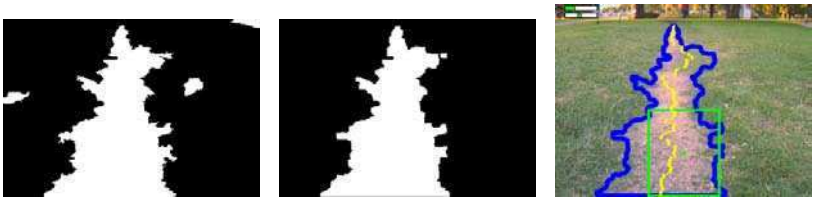
As mentioned above, we assume that the robot moves on a flat terrain. However, the path the robot has to follow could be uneven causing the camera to point up and down, thus the horizon line may not always be at the same height. Hence, a fast horizon detection algorithm is used to find the area of the image where the horizon line is located. In [140] the authors propose an image-based horizon detection algorithm which rests on two basic assumptions: 1) the horizon line will appear approximately as a straight line in the

(a) Input image (b) Cropped & blured (c) Segment Map



(d) Hue per pixel (e) Mean hue per super-pixel (f) Value per pixel

(g) Mean value per super-pixel (h) Saturation per pixel (i) Mean saturation per super-pixel



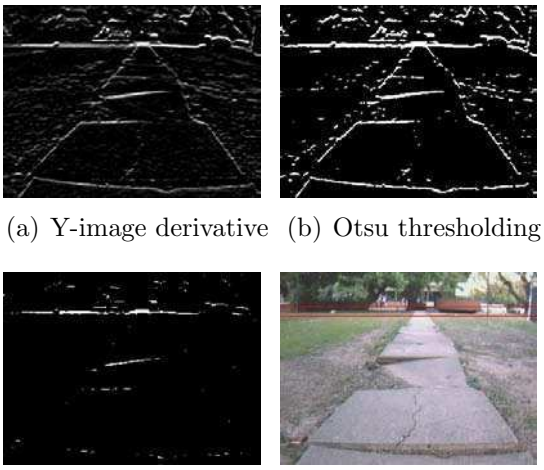
(j) Super-pixel classification as binary mask (k) Filtered binary mask (l) Path contour

Figure 4.2: Pipeline description: (a) input image as acquired from the camera, (b) cropped image below automatically detected horizon, (a) segment map. For each segment, mean and covariance are computed:(e), (i) and (g) show segment hue, saturation and value means respectively (d), (h) and(f) show pixel hue, saturation and value for reference. (j) binary mask obtained from classification using ground models from ROI. (k) morphological opening filter is applied in order to 'close' small gaps in the binary mask. (d) path contour extracted from processed binary mask and middle points computed, on top-left linear and angular speeds values obtained by control law.

image; and 2) the horizon line will separate the image into two regions that have different appearance, i.e. sky pixels will look more like other sky pixels and less like ground pixels, and vice versa [141]. The same assumptions are used in other image-based horizon detection algorithms [142] and are valid for the case of autonomous vehicles that navigate through open areas or for Unmanned Aerial Vehicles (UAV's). However, the second hypothesis may not be true for unstructured outdoor environments, where it is common that pixels above the horizon line correspond to objects, like plants, trees, cars, buildings or even people rather than only sky, and can not then be represented with the same model.

Thus, instead of using the second hypothesis, in this work we assume that the horizon line will be more or less parallel to the bottom of the image, which is valid for robots that operate on flat or small slope terrains. Based on this assumption we define the horizon as the area of the image where the greatest change in pixel intensity over the Y axis direction is detected. We use the Sobel filter as a discrete differentiation operator to compute an approximation of the gradient of the image intensity in Y axis direction. Then, we apply the Otsu thresholding method (OTM) to the image derivative in order to segment the image into two classes and therefore obtain a binary image. OTM finds the best threshold which maximizes interclass variance and minimizes intraclass variance. This is appropriate for path detection, because OTM overcomes the negative impacts caused by environmental variation, without user assistance. Moreover, it's low computational complexity makes it suitable for real-time applications and it still remains one of the most referenced thresholding methods [143]. There are some previous works that use OTM for horizon detection [142], but in this case we apply it to the image derivative.

Once we have a binary image, we apply an erosion filter to reduce noise and remove thin lines. Afterwards, we divide the image into a number of sub-images consisting of horizontal stripes of the same width. Then, we compute the number of pixels above the Otsu threshold for each sub-image, which can be thought of as a histogram of pixels that belong to the horizon, following the idea of [142]. For our tests, we use 10 sub-images. The sub-image with the highest amount of foreground pixels is where the horizon is expected to be. To obtain better results we compute the histogram two times: in the second iteration we start the division of the image at an offset corresponding to half the height of the sub-images, in order to overlap the first iteration. This is very useful for cases where the horizon is in between two sub-images. In this case, we can find a better sub-image candidate containing the horizon with the second computed histogram. In order to choose between the two sub-images detected during each pass, the candidate sub-image with the highest amount of foreground pixels is chosen as the winner. Because the goal of this algorithm is to detect where the horizon is to crop the image for the next steps of the method, and not the horizon itself, it finishes here. If the focus were placed on



(a) Y-image derivative (b) Otsu thresholding

(c) Erode (d) Horizon detection

Figure 4.3: Pipeline description of the horizon detection algorithm:(a) Sobel operator is first applied to the input image to obtain Y-image derivative, (b) Otsu thresholding is used to transform Y-image derivative to binary, (c) an erosion filter is applied in order to reduce noise and remove thin lines, (d) a histogram is computed to find the sub-image where the horizon line is located.

detecting the horizon, then the Hough Transform could be applied to extract the horizon line from inside the selected sub-image. Figure 4.3 resumes the horizon finding algorithm.

4.5 Segmentation Methods

After the horizon is detected and the image is cropped, median-blur filter is applied in order to reduce noise without smoothing edges. The next step is to obtain the segmentation map of the resulting image.

As it will be shown, image segmentation is the most time-consuming step of the whole navigation method. In order to satisfy real-time and on-board execution constraints, several segmentation methods were considered and tested. In particular, the Graph-based segmentation algorithm and Quick Shift segmentation algorithm happened to be the most adequate for the two computing platform utilized, This platform consists of a standard notebook mounted on the robot and a low-power GPU processor on-board the robot.

4.5.1 Graph-Based segmentation

In [138] an efficient graph-based segmentation method is presented. In general terms, the algorithm first constructs a fully connected graph where each node

corresponds to a pixel in the image. Pixel intensities between neighbors are analyzed and edges are broken whenever a threshold is exceeded. The resulting unconnected sub-graphs define the segments.

The method works as follows. An undirected graph $G = (V, E)$ is defined, where V is the set of vertices (pixels) and E is the set of edges. Each edge $e_{i,j} \in E$ has an associated weight w_{ij}, which indicates the dissimilarity between vertices v_i and v_j. In image segmentation, this weight is obtained by a difference in pixel intensity, color, location, etc. The segmentation S can be defined as a partition of V into connected components $C \in S$ of a graph $G' = (V, E')$, where $E' \subseteq E$. The final result of the segmention is such that edges between two vertices in the same component have relatively low weights, and edges between vertices in different components have higher weights. Pseudo-code is presented in listing 2.

Algorithm 2 Graph-based image segmentation algorithm in pseudo-code.

```
Sort E into π = (e_1, ..., e_m), by non-decreasing edge weight.
Define initial segmentation S_0, such that C_i = v_i
for q = 1 to m do
    Define v_i, v_j as the vertices connected by edge e_q.
    Construct S_q as follows: if v_i and v_j are in disjoint components of S_{q-1}
      and w(e_q) is small compared to the internal difference of both those
      components, then merge the two components.
end for
```

As demonstrated by the authors, the proposed algorithm is found to have $O(m\ log(m))$ complexity for the case of non-integer weights.

4.5.2 Quick shift segmentation

Quick shift[144] is an example of a non-parametric mode-seeking algorithm, which aims to estimate an unknown probability density function. Density estimation is performed by associating data points to modes of the underlying probability density function.

One commonly used approach for the estimation is to use a Parzen-window for the density estimation[145]. Formally, given N data points $x_1, \ldots, x_N \in \chi = \mathbb{R}^d$, the Parzen-window approach estimates the probability as:

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} k(x - x_i), \tag{4.1}$$

where $k(x)$ is commonly referred to as the *kernel*, which is generally written as a Gaussian. The mode-seeking algorithm evolves data points x_i towards

a mode of $p(x_i)$, by means of gradient ascent over the kernel. All points associated to the same mode form a cluster.

There are several mode-seeking methods that differ in the strategy used to evolve those data points towards a mode. Quick shift is actually an improvement over Medoid shift [146] which, in turn, is an improvement over the original Mean shift [147] method. Medoid shift simplifies this evolution of the datapoints by restricting the point trajectories to only move over the data points x_i themselves. The downside of this approach is mainly its slow speed in practice[144]. Finally, Quick shift simplifies trajectories even further by not analyzing the gradient and simply moving data points to their nearest neighbor for which there is an increment in the density $p(x)$.

In [144] Quick shift is applied to the problem of image segmentation and it is shown that their proposed method is considerably faster than Mean shift, and marginally faster than Medoid shift.

Algorithm 3 Quick shift image segmentation algorithm in pseudo-code.

function COMPUTEDENSITY($image$)
  for x in all pixels of $image$ **do**
    $P[x] \leftarrow 0$
    for y in all pixels less than 3σ away from x **do**
      $P[x] \leftarrow P[x] + e^{\frac{-(f[x]-f[y])^2}{2\sigma^2}}$
    end for
  end for
  return P
end function

function LINKNEIGHBORS($image, P$)
  for x in all pixels of $image$ **do**
    for y in all pixels less than τ away from x **do**
      if ($P[y] > P[x]$ and $distance(x, y)$
        is smallest among all y) **then**
        $d[x] \leftarrow distance(x, y)$
        $parent[x] \leftarrow y$
      end if
    end for
  end for
  return d, $parent$
end function

The first step of the Quick shift algorithm involves the computation of the density for each pixel, by means of analyzing a local neighborhood inside which contributes to the density are the most significative. The second step links every pixel to its nearest neighbor with higher density. Pseudo-code is presented in listing 3.

4.5.3 Quick shift on the GPU

GPU's (Graphics Processing Units) have lately gained considerable popularity as cheap, powerful programmable general purpose processors outside their original application domain. Recent models are able to sustain over hundreds of GFLOPs and due to their highly parallel architecture, GPUs are attractive platforms for intensive data-parallel algorithms. This type of hardware is therefore very well suited for on-board mobile robots with vision-based perception. Although general-purpose computing on GPU (GPGPU) has been an active area of research for decades, the introduction of Compute Unified Device Architecture (CUDA) and CTM has finally brought it within reach of a broader community, giving programmers access to dedicated application programming interfaces (APIs), software development kits (SDKs) and GPU-enabled C programming language variants.

In [148] an implementation of the Quick shift algorithm for execution on GPU is presented. Due to the independence in the computation of the density between different pixels, the authors exploit the parallel execution capabilities of the GPU for this computation. By using several features of this kind of computing platforms, the cost associated to the type of access pattern associated to a parallel implementation of Quick shift is greatly reduced. Since the computation of the density for a single pixel requires accessing a local neighborhood of pixel densities, the redundant access of neighbors is managed by using the texture memory space of the GPU which includes efficient caching.

Compared to a CPU implementation, the proposed algorithm implementation is between 10 to 50 times faster when running on medium-range hardware. For the case of 256×256 pixel resolution images, the GPU implementation works at 10 Hz.

In this work, the Quick shift algorithm running on a GPU was chosen due to its speed and simplicity. However, the hardware utilized for experiments by the authors is still more powerful than the hardware found on low-power embedded platforms commonly present on mobile robots. Therefore, in this work several simple optimizations were included in the originally available source-code [149] which implements the GPU version of the Quick shift algorithm.

The first optimizations that were introduced consist of carefully tuning the number of concurrent threads executed and the registers required for code

compilation (which affects the efficiency of the thread scheduling and thus the parallelization level), and also taking into account the capabilities of the specific card to be used. The second main optimization that was introduced involves a simpler handling of out-of-bound accesses which arise when searching the neighborhood of pixels near the edges of the image. In the original implementation these were avoided explicitly, while in our case, the *clamping mode* of the texture memory is used. Here, these accesses simply return the nearest valid pixel in the image. By introducing this change, the code can be simpler and more efficient. Finally, memory accesses in general were reduced by delaying them up to the point where there was certainty that they were required.

These optimizations reduce computation to the point of allowing real-time execution on the robot embedded GPU. The speedup obtained when compared to the original implementation running on the same low-power GPU is around 4.8 times (see detailed results in Section 5.5).

4.6 Classification Method

To achieve a robust classification of the segments, a probabilistic approach is used, based on [140].

In abstract terms, the classification step aims to determine if a population sample (a segment), modeled by a Gaussian probability distribution $\mathcal{N}(\mu, \Sigma)$, with mean vector μ and covariance matrix Σ, represents an instance of a more general model of the 'navigable path' class or not. This navigable path class is represented in turn not by a Gaussian, but by a Mixture-of-Gaussians (MoG) model. However, unlike [140], in this work the gaussian elements of this mixture model are readily available and therefore the global mixture model is not explicitly computed.

In the following sections, the classification method is presented in detail.

4.6.1 Segment Model Computation

Alternatively to [140] we use HSV color space to compute the segment model. The reason for using HSV color space is that, in contrast of the RGB color space, is that the chrominance and luminance information are maintained in separate channels. This provides better control when selecting thresholds over each HSV channel and also provides some degree of invariance between chrominance and luminance.

The segment model consists of mean and covariance of a Gaussian probability distribution $\mathcal{N}(\mu, \Sigma)$. This model should be computed from segment

pixels in the HSV representation. However, computing the mean of Hue values from a sample is, in principle, ill-posed since this channel is actually a circular measure (can be interpreted as an angle). The solution to this problem is to compute first the RGB segment model and then to use this result to obtain the HSV segment model.

Computing the segment model in the RGB color space is straightforward: given a segment composed of N pixels $P^{RGB}_{i=1...N}$ represented as three-dimensional vectors in the RGB color-space, with mean μ^{RGB} and covariance Σ^{RGB} of this segment are computed as follows:

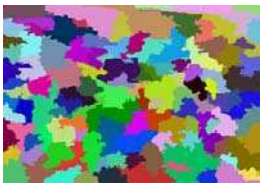
$$\begin{aligned} \mu^{RGB} &= \sum_i^N \frac{P_i^{RGB}}{N} \\ \Sigma^{RGB} &= \frac{(P_i^{RGB}-\mu^{RGB})(P_i^{RGB}-\mu^{RGB})^T}{N-1} \end{aligned} \tag{4.2}$$

Now, the mean μ^{RGB} is converted into HSV to obtain μ^{HSV}. Finally, in order to compute the HSV covariance matrix Σ^{HSV}, the distance between a pixel P_i^{HSV} and the mean μ^{HSV} is restricted to angles less than 180° in HSV space. It should be noted that this approach requires the original RGB representation of the image, along with the computed HSV version. However, the RGB covariance matrix Σ^{RGB} is actually not required.

4.6.2 Path Class Model Computation

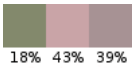
In order to compute a model for the path class, a rectangular region of interest (ROI) corresponding to the area directly in front of the robot is used as an example of the navigable path class appearance. From this ROI a model is extracted. However, this area may contain very dissimilar information due to textures (e.g. tiled path) or outliers (e.g grass on the side of the cement road). Therefore, this area is represented not by a Gaussian, but by a Mixture of Gaussians (MoG) model. Given that the image is already segmented and that each segment is modeled with a Gaussian distribution, the elements of the mixture model are already computed. Figure 4.4 shows an example of the segment map with the mean μ^{RGB} for each segment, the ROI corresponding to the area in front of the robot and samples of the mean μ^{RGB} for each segment that overlaps the ROI, with its coverage percentage.

The classification step starts by computing the intersection of the segments of the image and the rectangular ROI. Then, segments in this intersection are re-grouped by iteratively joining similar models. This similarity is defined in terms of the Mahalanobis distance, defined in this case for two Gaussian distributions. Two distributions are said to be similar when:

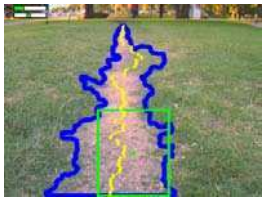


(a) Segment Map

(b) Segments with μ^{RGB}



(c) μ^{RGB} samples from segments that overlap ROI with its coverage percentage



(d) ROI, extracted path contour

Figure 4.4: Classification method: (a) segment map. For each segment, mean and covariance are computed, (b), (c) μ^{RGB} samples from segments that overlap ROI with coverage percentage (d) ROI, path contour extracted, on top-left linear and angular speeds values obtained by control law.

$$(\mu_1 - \mu_2)^T(\Sigma_1 + \Sigma_2)^{-1}(\mu_1 - \mu_2) \leq 1 \tag{4.3}$$

The process of joining similar segments inside this ROI works iteratively, by merging two segment's mean and covariance matrices into one. On each step, for each segment inside this region, among all other segments that satisfy equation (4.3), the nearest neighbor in Mahalanobis space is chosen for merging. This merging by pairs continues until no more segments can be merged. In order to merge two segments with means μ_1, μ_2, covariance matrices Σ_1, Σ_2 and number of pixels N_1, N_2, a new segment is obtained by:

$$\begin{aligned} N_f &= N_1 + N_2 \\ \mu_f &= (\mu_1 N_1 + \mu_2 N_2)/(N_1 + N_2) \\ \Sigma_f &= (\Sigma_1 N_1 + \Sigma_2 N_2)/(N_1 + N_2) \end{aligned}$$

With this minimal set of Gaussian models, each segment in the image can now be classified as belonging to the path or non-path classes by comparing them to all elements of this set. Only elements which cover the ROI by more than a specified threshold c (coverage) are considered. In this way, small outliers inside the ROI can be ignored. Again, the notion of similarity is defined

as in equation (4.3). Since the ROI is represented by a mixture of Gaussians (MoG), if the path is built of different colors or textures, for example black and white tiles, the classification method still works and all the tiles will be labeled as belonging to the path class. The same happens in the presence of shadows. On the other hand, if there is a small forbidden region (e.g. a piece of grass) on the middle of the path, this segment will have a small coverage percentage and hence it will not affect the classification. Finally, if an obstacle (sufficiently distinguishable) appears in the middle of the path, it will be classified as non-path, affecting the computation of middle points and therefore the robot will steer away from the obstacle and avoid collisions. In Figure 4.5 it can be seen that, when a person stands in front of the robot the algorithm classifies the person as non-path and the robot follows a trajectory to circumvent him.

In equation (4.3) the sum of the covariance matrices is post-processed in order to include the notion of minimum covariance. These minimums are added in the computation as a way to increase the threshold used for this classifier from (4.3), but allowing to affect each channel of the HSV color space differently. For the case of re-grouping similar segments, this permits to be more permissive by relaxing the notion of similarity. On the other hand, when classifying segments, these thresholds account for the variance which exists in the complete visible path, as opposed to only the region inside the ROI. This issue appears frequently since the color of the path changes smoothly towards the vanishing point and therefore the region inside the ROI may not accurately represent the appearance of the complete path.

In order to apply these minimums to the sum of the covariance matrices, an eigenvalue decomposition of the summed matrix is first obtained:

$$\Sigma_{sum} = VDV^T \tag{4.4}$$

where V is the matrix of eigenvectors and D the diagonal matrix of eigenvalues. Then, D is modified as

$$D'_i = \begin{cases} D_i & D_i > T_i \\ T_i & \text{else} \end{cases}$$

where T is a diagonal matrix of minimum values. Finally, a new covariance matrix is recomposed using equation (4.4) using V and D'.

It should be noted that the thresholds used for grouping path segments do not generally depend on the environment and can be preset. On the other hand, thresholds used for classification may need to be adjusted when high color variance exists in the complete path.

4.7 Path Contour Extraction

The previous classification step produces a binary mask which distinguishes path from non-path pixels. The goal of the following step is to extract a contour of the path within the captured image. With this contour, a simpler path representation (as middle points) can be used in order to apply a control law which will maintain the robot centered.

The binary mask, however, may contain several unconnected patches of pixels detected as path, since areas of similar appearance may exist outside but nearby the navigable area. Therefore, the first task is to identify the correct patch. Two possibilities were analyzed: to find the patch which includes the center point of the rectangular ROI or to extract the contour with the largest perimeter. Experiments have indicated that the latter approach is more robust.

Once this contour is extracted, a new binary mask consisting only of the selected patch is generated. This second mask is then processed with a morphological opening operation (an erosion followed by a dilation) with the main purpose of removing peninsulas from the main patch, which may appear in naturally demarcated paths and should not be included in the following steps. While the erosion operation may disconnect these patches, the area of all path regions will be reduced. Therefore, by dilating the image afterwards, all path areas are again expanded without reconnecting them.

The final path contour is again extracted from the processed binary mask.

4.8 Motion control

From the previously determined contour, a path center trajectory is estimated in order to maintain the robot centered and away from path edges. First, by going row-by-row in the image, the middle point for the current row is obtained by subtracting the leftmost and rightmost pixel's horizontal positions of the path contour.

The list of middle points is used to compute angular and linear speeds with a simple yet effective control-law, based on previous work [150]. From the list of n horizontal values x_i of the i-th middle point of the detected path region, angular speed $\dot{\omega}$ and linear speed $\dot{x}$ are computed as follows:

$$\dot{\omega} = \alpha \sum_{i}^{n} \left(x_i - \frac{w}{2} \right)$$

$$\dot{x} = \beta n - |\omega|$$

where w is the width of the image and α and β are motion constants.

The effects of this control law are such that the robot will turn in the direction where there is the highest deviation, in average, from middle points with respect to a vertical line passing through the image center. This line can be assumed to be the position of the camera, which is mounted centered on the robot. Therefore, whenever a turn in the path or an obstacle is present, the robot will turn to remain inside the path and avoid the obstacle. The linear speed of the robot is inversely proportional to the angular speed determined by the previous computation. This has the effect of slowing down the robot whenever it has to take a sharp turn.

4.9 Experimental results

The proposed method was tested experimentally in several aspects. In terms of performance, computation time was measured over two distinct platforms (a modern laptop and an embedded hardware on-board robot) for the individual segmentation step and for an iteration of the complete method. For qualitative analysis, the road detection capability of the system was evaluated using offline execution over previously captured images, as acquired by the robot's camera during a human-guided step. Finally, online testing was performed by executing the proposed algorithm on-board the robot in real-time in order to assess the closed-loop behavior.

The robot used for online experiments (and some offline dataset acquisition) was the ExaBot[27] (Figure 4.1), which features a differential-drive motion and supports different sensor configurations depending on the needs. For this work, the robot was equipped with an AT3IONT Mini-ITX computer, featuring a dual core Intel Atom 330 processor with an embedded ION nVidia graphics card. This embedded GPU is CUDA enabled, allowing it to be used as a GPGPU platform, with 16 cores running at 1.1 GHz and with 256 MB of RAM. On the other hand, as a reference, a modern Laptop with an Intel Core i5 at 2.30 GHz and 4 GB of RAM was also used for performance measurements.

A firewire camera was used for acquiring images (model 21F04, from *Imaging Source*) with a $3.5 - 8mm$ zoom lens, set at its widest focal length. While the camera is capable of capturing images at 640×480 px images at 15 fps, a smaller resolution of 320×240 px (at 30 fps) was chosen since it was enough for proper road detection. This smaller resolution also decreases computation times.

4.9.1 Offline Testing

In order to assess the quality of the algorithm in terms of its ability to perform path detection, the proposed method was executed on more than 1000 images, belonging to a dataset of various environments and seasonal conditions. Images were obtained from different sources: some of them were acquired with the camera mounted on the ExaBot robot (altitude from ground: 40 cm) while others were acquired by a camera mounted on a Pioneer 3AT robot (altitude from ground: 70 cm). These datasets depict different situations with varying degrees of difficulty in terms of road distinctiveness from surrounding areas, road shape, texture and color, under many lighting conditions including shadows and reflections and during different seasons.

4.9.2 Online Testing

For the purpose of analyzing the stability of the navigation when the algorithm is executed online with a stream of images acquired on real-time by the camera, the robot was placed on outdoor roads and positioned in different starting configurations, some of which consisted of extreme cases which could not be reached without manually displacing the robot. As an example, a series of successive frames are presented on Figure 4.6. The robot is initially placed at one side of the road (Figure 4.6(a)), pointing away from it. After enabling robot control, the robot soon turns towards the road (Figures 4.6(b)-4.6(d)) advances and then turns again to remain on course (Figures 4.6(e)-4.6(f)).

Since real-world testing is a time costly process, system behavior has been first tested thoroughly over off-line datasets and then simple control law testing was performed to ensure stability of robot motion.

4.9.3 Segmentation Methods Performance

In this work, several image segmentation methods were considered in order to meet the real-time constraints required for on-board execution on embedded hardware. In this section, the execution time of these algorithms is presented, measured on different platforms. While each method utilizes a different set of parameters, the corresponding values where chosen in each case by maximizing execution speed. The size and the number of segments (as expected, these quantities are inversely proportional to each other) are important for the classification quality and computation speed of the navigation algorithm as a whole. The differences obtained in the resulting segmentations for the chosen set of parameter values, were negligible.

In Figure 4.8, mean execution speeds (over 32 iterations) are presented corresponding to different segmentation algorithms, executing platforms and

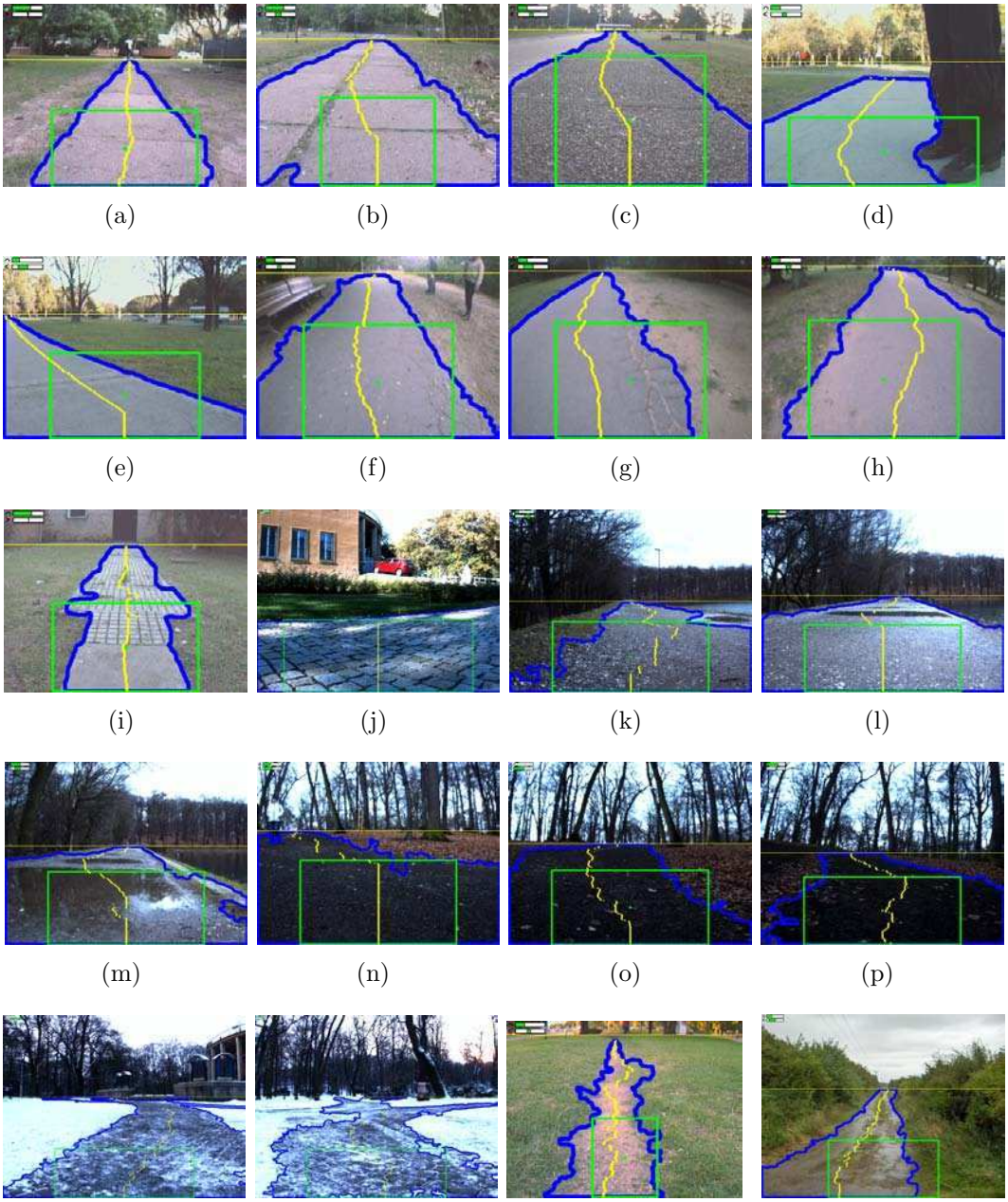
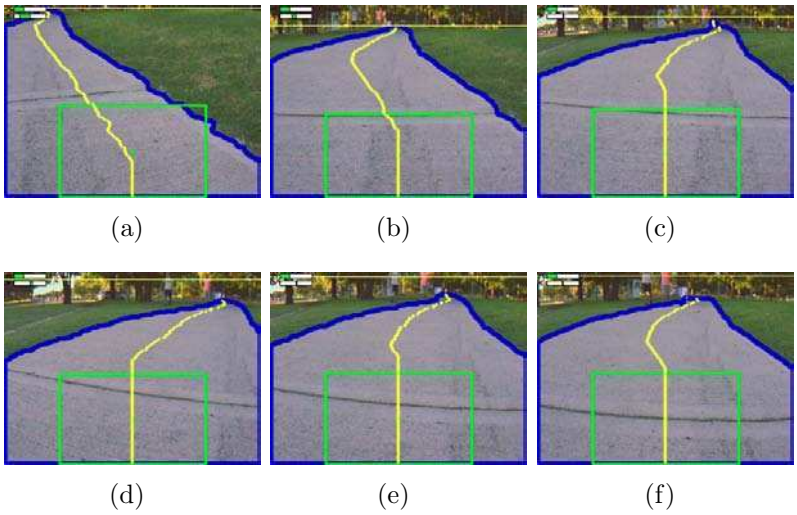


Figure 4.5: Example final images processed offline, as obtained from several datasets: the green rectangle represents the ROI used for extracting road appearance samples, the yellow line corresponds to the automatically detected horizon, the blue contour delimitates the detected ground region, yellow points correspond to road middle-points from which control law is computed, small bars in top-left show the control law output for linear (top bar) and angular speed (bottom bar), going from 0 to 1 and from -1 to 1, respectively.



(a) (b) (c)

(d) (e) (f)

Figure 4.6: Example of online testing: series of successive frames where robot was started from a deviated position and ended with robot centered in road

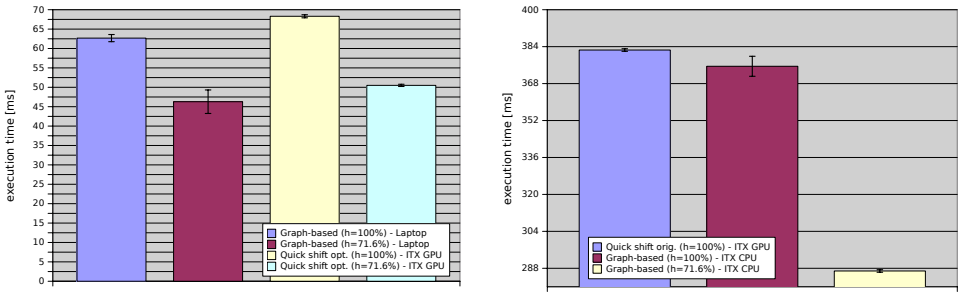
Figure 4.7: Image used for image segmentation algorithm evaluations

relevant parameters. The algorithms were executed over a test image captured during outdoor experiments (see Figure 4.7).

On Figure 4.8(a) acceptable timings are presented, whereas on Figure 4.8(b) the execution speeds presented do not permit real-time execution and stable control of the robot.

The first set of measurements (Figure 4.8(a)) include the execution of the Graph-based segmentation algorithm on the Laptop's processor and the optimized Quick shift algorithm running on the GPU of the Mini-ITX computer. These combinations of algorithms and executing platform were found to be the fastest ones, being also within real-time constraints. For both of these cases, the complete image was processed (horizon set to 100%) and only the relevant part (horizon set to 71.6%) according to the test image.

The optimal tradeoff between total number and individual size of segments was found to be around 190 segments. For the case of the Graph-based al-



(a) Optimal choices of segmentation algorithm and executing platform

(b) Other combinations of segmentation algorithm and executing platform

Figure 4.8: Computation time for the segmentation step, for different algorithms and executing platforms

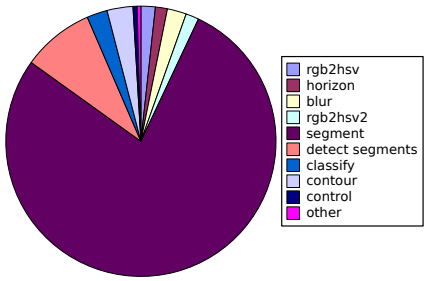
gorithm the threshold for segment splitting was $t = 140$. For the Quick shift algorithm, the thresholds used were $\sigma = 2$ and $\tau = 8$.

For completeness, a second batch of measurements was performed (Figure 4.8(b)), comparing the previous measurements to the execution of the unmodified Quick shift algorithm running on the same GPU and the Graph-based algorithm running on the Mini-ITX dual core CPU. On these conditions, the computation times of the segmentation step alone already exceed the tolerance for stable control of the mobile robot, which should be in the order of 4 Hz for the whole algorithm.

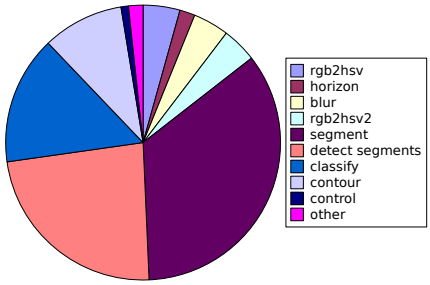
Finally, by comparing the execution times (Figures 4.8(a) and 4.8(b)) between the original and optimized versions of the Quick shift segmentation algorithm running on GPU, it can be seen that with these optimizations a speedup of approximately 5.6× is achieved.

4.9.4 Complete Algorithm Performance

The proposed algorithm can be decomposed into several steps. In this section, the time consumed by each step is presented. In Figure 4.9 the computational time of each step, relative to the total iteration, is presented. In Figure 4.10 the absolute computational time consumed by each step is shown. In both cases, two series of timings were measured, corresponding to the segmentation algorithm and parameters choice which maximize execution speed, for the two computing platforms considered (Laptop and embedded GPU). These time measurements correspond to the mean value of $N = 66$ repetitions over the same input image. Standard deviation is presented on Figure 4.10 for each step. The total mean time consumed by one iteration of the algorithm was 59.9841 ms (std. dev. 1.2011 ms) when executed on the Laptop, and 179.1447 ms (std. dev. 1.744 ms) when executed on the GPU.



(a) Computation time of algorithm as executed on the Core i5 processor present on the Laptop



(b) Computation time of algorithm as executed on the GPU present on the Mini-ITX embedded computer

Figure 4.9: Execution time for each step relative to the whole algorithm executed on a Laptop ((a)) and the Mini-ITX GPU ((a)): rgb2hsv: conversion of original image to HSV color-space, blur: median-blur applied to input image, horizon: horizon detection, rgb2hsv2: conversion to HSV of smoothed image, segment: image segmentation algorithm (Graph-based on CPU, Quick shift on GPU), detect segments: construction of map of labeled pixels and computation of mean and covariance for each segment, classify: ground model construction and individual segment classification, contour: binary mask computation from classified segments and road contour, control: control law extraction from road middle-points

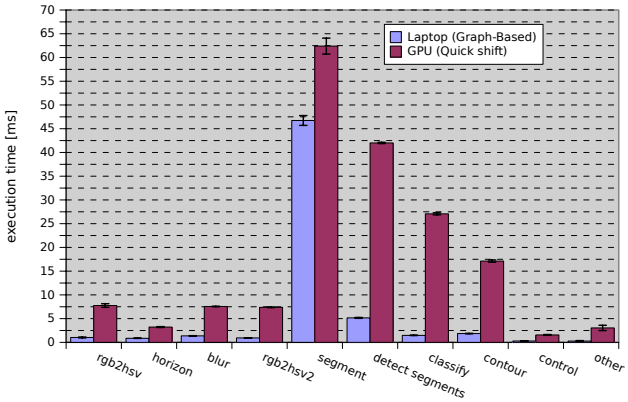


Figure 4.10: Absolute execution time for each step of the algorithm, in both platforms

4.10 Conclusions

In this Chapter we presented a novel autonomous navigation method for outdoor environments. This method allows a mobile robot, equipped only with a monocular camera, to follow structured or semi-structured paths. It does not require training phase, camera calibration or any *a priori* given knowledge or map of the environment to maximize its adaptability. The core of the method is based on segmenting images and classifying each segment to infer a contour of navigable space. Middle points of this contour are extracted to define a trajectory for the robot in the image. Finally, a simple yet stable control law sets the linear and angular speeds in order to guide the robot through the middle of the detected path. To achieve real-time response and reactive behaviour for the robot, two image segmentation algorithms are tested: one implemented on CPU and the other implemented on a low-power on-board embedded GPU. The validity of our approach has been verified with an image dataset of various outdoor paths as well as with the mobile robot ExaBot.

The presented method was tested only in outdoor environments, but it could also be applied to indoor environments, for example to traverse corridors. The method works under the assumption that the robot operates in an environment characterized by a set of regions connected by a network of paths. However, it does not address the problem of how to navigate in open areas where there are no distinguishable paths or how to cross intersections of two or more paths. For this purpose other method based on visual landmarks will be presented in the next Chapter.

Chapter 5

Landmark-based visual navigation

This Chapter describes a landmark-based monocular navigation method and its implementation for the ExaBot. This method uses the map-and-replay paradigm for navigation in indoor and outdoor unstructured environments.

5.1 Introduction

As we saw in the previous Chapter, it is possible to get a navigation method that does not require to build a map of the environment or localize the robot on it. However, if there is no path, road or other bounded navigable area in the environment to go through, we still have to specify the desired trajectory for the robot somehow. One possibility is to specify the trajectory as a sequence of waypoints in a known reference frame. This requires an external localization system (such as a GPS) or a map of the environment. In turn, this map can be given in advance (*a priori*) to the robot, or it can be simultaneously created as the robot localizes itself on it (SLAM). Another possibility is to specify the trajectory as a series of sensory measurements the robot perceives along the desired path. Typical examples are the systems where the path is given as a sequence of images. In this case, the problem of autonomous navigation is divided into two phases, what is known as map-and-replay technique. With this approach, the robot first traverses the desired path during a human-guided training phase so it can build a representation of the environment, i.e. a map. In the second phase, the robot can use this map to navigate autonomously through the learned path. That is why this approach is also known as teach-and-replay.

In this Chapter, we will present a landmark-based monocular navigation method based on the one proposed by Tomáš Krajník in [49] and later detailed

in his PhD thesis [151]. It uses the map-and-replay technique. The built map contains visual natural landmark information about the environment that is then used during the autonomous navigation phase. The basic idea is to utilize image features as visual landmarks to correct the robot's heading, leaving distance measurements to the odometry. The heading correction itself can suppress the odometric error and prevent the overall position error from diverging. The presented method is robust and easy to implement and does not require sensor calibration or structured environment, and its computational complexity is independent of the environment size. The aforementioned properties of the method allow even low-cost robots to effectively navigate in large indoor and outdoor environments. Experiments with ExaBot show these benefits in practice. The original method uses a digital compass sensor to measure the robot turns. In the presented implementation only wheel shaft encoders are used, thus the turns of the robot are only measured and performed by means of odometry, which makes the method even simpler.

Furthermore, and most importantly, the original version of the method uses SURF (Speeded Up Robust Features) as visual landmarks. As a result of a thorough evaluation of a variety of image feature detector-descriptor schemes explained in the next Chapter, we propose to use STAR based on CenSurE (Center Surround Extremas) to detect image features and BRIEF (Binary Robust Independent Elementary Features) to describe them. Using STAR/BRIEF instead of SURF results in a great improvement to the method.

The rest of the Chapter is organized as follows: Section 5.2 mentions some related works, Section 5.3 describes the method in detail, Section 5.4 shows the convergence of the method, Section 5.5 presents some results for indoor and outdoor environments and finally Section 5.6 concludes this Chapter.

5.2 Related work

There are lots of works that use teach-and-replay technique to achieve autonomous navigation. In the article [152], a monocular camera is carried through an environment and a video is recorded. The recorded video is then processed (in a matter of several minutes) and subsequently used to guide a mobile robot along the same trajectory. More similar to the method described in this chapter, authors of papers [153, 114] present an even simpler form of navigation in a learned map. Their method utilizes a map consisting of salient image features recorded during a teleoperated drive. The map is divided into several conjoined segments, each one associated with a set of visual features detected along it and a milestone image indicating the segment end. When a robot navigates a segment, its steering commands are calculated from positions of the currently recognized features and also from the already mapped features. The robot using this method moves forward with a constant speed

and steers right or left with a constant velocity or does not turn at all. In paper [153], the end of the segment was detected by means of comparing the milestone image with the current view. An improved version [114] of the qualitative navigation uses a more sophisticated method to determine the segment end. However, the authors mention difficulties to properly detect the segment end.

5.3 The method in detail

5.3.1 Image features as visual landmarks

Since image features are considered as visual landmarks, the feature detector and descriptor algorithms are a critical component of the navigation method. Image features must provide enough information to steer the robot in a correct direction. Furthermore, they should be robust to real world conditions, i.e. changing illumination, scale, viewpoint and partial occlusions and of course its performance must allow real-time operation.

In the original version of the method the author decided to use Speeded Up Robust Features (SURF) to identify visual landmarks in the image. SURF provides both a feature detector to extract salient points from the image called keypoints (or interest points) and a feature descriptor to provide a numerical representation of the keypoints. The SURF method is reported to perform better than most SIFT implementations in terms of speed and robustness to viewpoint and illumination change.

As part of this Thesis, further evaluation of a variety of image features detector-descriptor schemes for visual navigation was performed (see Chapter 6). SIFT, SURF, CenSurE, BRIEF, ORB and BRISK methods were tested using a long-term robot navigation data set. The goal of these off-line experiments was to determine which image feature detector-descriptor scheme is the best choice for visual navigation in terms of performance, robustness and repeatability. As a result of this analysis we concluded that the CenSurE detector using STAR implementation and BRIEF as descriptor is the best configuration to achieve the best detector-descriptor schema for robot navigation. On the one hand, the STAR detector gives better results to detect landmarks than SURF, because keypoints extracted with STAR are more salient and thus, also more stable than keypoints extracted with SURF. On the other hand, the BRIEF descriptor gives the shortest form to describe the landmarks and also a faster way to compare them. Instead of using the Euclidean distance to compare descriptors, as it is done with SURF, BRIEF descriptors can be compared using Hamming distance, which can be implemented extremely fast on modern CPUs that provide a specific instruction to perform a XOR or bit count operation, as is the case in the latest SSE instruction set. For

further information see Chapter 6.

Typical images from indoor and outdoor environment with highlighted STAR feature positions are shown in Figure 5.1.

(a) Typical indoor scene. (b) Typical outdoor scene.

Figure 5.1: Robot camera view with detected landmarks (red circles) using STAR detector algorithm.

5.3.2 The map

The key component of the proposed navigation procedure is a map. The method uses a topological map, i.e. it represents the environment as a directed graph. Each map edge has its own local landmark map, consisting of salient features detected in the image captured by the robot's forward looking camera. The edge description also contains its length and orientation. The map is created during a teleoperated run, in which the robot operator guides the robot in a turn-move manner. Immediately after mapping, the operator can start the autonomous navigation.

The local segment landmark map does not contain information about real spatial positions of the landmarks. Instead, it states in which part of the segment the landmarks were visible and what their image coordinates were. This information is sufficient to compute image coordinates of the landmarks for particular robot positions.

As one can see from a particular example of the map on Table 5.1, each segment is described by its length s, azimuth α and a set of detected landmarks L. A landmark $l \in L$ is described by a tuple $(\mathbf{e}, k, \mathbf{u}, \mathbf{v}, f, g)$, where $\mathbf{e}$ is its BRIEF descriptor and k indicates the number of images, in which the feature was detected. Vectors $\mathbf{u}$ and $\mathbf{v}$ denote positions of the feature in the captured image at the moment of its first and last detection and f and g are distances of the robot from the segment start at these moments.

Table 5.1: Local map in a text file

Record	value	meaning
Initial turn and length:	2.13, 7.03, 0.785	α,s
Landmark 0:		
First position:	760.74, 163.29	$\mathbf{u_{l_0}}$
Last position:	894.58, 54.44	$\mathbf{v_{l_0}}$
First and last visible distance:	0.00, 4.25	f_{l_0}, g_{l_0}
Descriptor:	1110101101101010...	$\mathbf{e_{l_0}}$
Landmark 1:		
First position:	593.32, 381.17	$\mathbf{u_{l_1}}$
Last position:	689.89, 377.23	$\mathbf{v_{l_1}}$
First and last visible distance:	0.00, 6.73	f_{l_1}, g_{l_1}
Descriptor:	0101010001010111...	$\mathbf{e_{l_1}}$

Referring to the memory usage for the map, the main issue is storing each descriptor for each landmark. Here is where using BRIEF descriptors becomes a substantial improvement over the original navigation method. Standard image feature detector and descriptor algorithms such as SIFT or SURF use 64 (or 128 in the extended version) double values to describe a keypoint (a feature). This meas 512 bytes for each descriptor (or 1024 in the extended version). Instead, BRIEF uses a string of bits that can be packed in 32 bytes (see Chapter 6). If we multiply this difference by the number of features in the map we can realise that hundreds of megabytes are saved for large-scale maps, significantly reducing the storage requirement of the method and making loading into memory faster when the robot has to use the map.

5.3.3 Learning Phase

In the learning phase, the robot is manually guided through the environment in a turn-move manner and creates a map consisting of several straight segments. The operator can either let the robot go forward or stop it and turn it in a desired direction. During forward movement, the robot tracks the features in the camera image and records their positions in a (local segment) landmark map. When stopped, the robot saves the landmark map and waits until the operator turns it in a desired direction and resumes forward movement to map another segment.

The procedure which creates a map of one segment is described in Algorithm 4. Before the robot starts to learn a segment, it resets its odometric counters. After that the robot starts to move forwards, and continuously tracks the detected features and puts them in the landmark map L until the operator requests to stop. The mapping algorithm can be described by means of manipulation with three sets of landmarks: the currently detected landmarks S, landmarks to be tracked T and map landmarks L. The set S changes each time a new picture is processed. For each currently tracked landmark $\boldsymbol{t_i}$ (from

Algorithm 4 Learn one segment

Output: (α, s, L) – associated data to segment, where s is traveled distance and L is set of landmarks, a landmark is tuple (k, e, u, v, f, g), where e is a feature descriptor, k is counter of feature detection, u and v is position of feature in the image (at the moment of first, resp. last occurrence), f and g denotes distance from segment start according to u, resp. v.

$L \leftarrow \emptyset$
$T \leftarrow \emptyset$
repeat
  $d \leftarrow$ current traveled distance from the start.
  $S \leftarrow$ extracted features $(u, e) \in S$, u position, e descriptor.
  for all $\boldsymbol{t_i} = (e_i, k_i, u_i, v_i, f_i, g_i) \in T$ **do**
    $(u_a, e_a) \leftarrow \text{argmin}\{||(e_i, e(s))||, s \in S\}$
    $(u_b, e_b) \leftarrow \text{argmin}\{||(e_i, e(s))||, s \in S - \{(u_a, e_a)\}\}$
    if $||(e_i, e_a)|| < 0.8 \times ||(e_i, e_b)||$ **then**
      $\boldsymbol{t_i} \leftarrow (e_i, k_i + 1, u_i, u_a, f_i, d)$
      $S \leftarrow S - \{(u_a, e_a)\}$
    end if
  end for
  for all $(u, e) \in S$ **do**
    $T \leftarrow T \cup \{(e, 1, u, u, d, d)\}$
  end for
until operator terminates learning mode
$s \leftarrow d$
$L \leftarrow L \cup T$

the set of tracked landmarks T), the two best matching features (measured by their descriptors' Hamming distance) from the set S are found. If one of the pairs is significantly more similar than the other, the tracked landmark $\boldsymbol{t_i}$ description (values k, v, g) are updated and the best matching feature is removed from S. If not, the landmark $\boldsymbol{t_i}$ is moved from the set T to the set L. The remaining features in the set S are added to the set of tracked landmarks T, their values of f,g are set to the current value of the traveled distance from the segment start and their counter of feature detections k is set to one. At the end of the segment, its description is saved and the operator can turn the robot to another direction and initiate mapping of another segment. A part of the file with segment description is shown in Table 5.1.

5.3.4 Autonomous Navigation Phase

The basic idea of the autonomous navigation phase is to retrieve a set of relevant landmarks from the map and to estimate their position in the current camera image. These landmarks are paired with the currently detected ones and the differences between the estimated and real positions are calculated. The mode of these differences is then used to correct robot heading.

To start the autonomous navigation the robot is placed at the start of the first segment, it loads the description of the first segment, resets its odometry, turns itself in the direction of the segment's direction, and starts moving forward. The navigation procedure is described in Algorithm 5.

Each time a new image is processed, the robot reads its odometric counter d and checks which landmarks in the set L have been seen at the same distance from the segment start, i.e. which landmarks have lower first detected distance f and higher last detected distance g than the value of the odometric counter d. The supposed position of these landmarks in the camera image is then computed from their u, v,f, g and d values by means of linear interpolation and they are put in the set of tracked landmarks T. Then, the pairing between the sets T and S is established similarly as in the mapping phase. A difference in horizontal image coordinates of the features is computed for each such pair. The mode of those differences is estimated by a histogram voting method. The mode is converted to a correction value of movement direction, which is reported to the robot motion control module. After the robot travels a distance equal to the length of a given segment, the next segment description is loaded and the procedure is repeated. During navigation, the robot displays current detected landmarks, estimated landmarks from the map, correspondences between them and the histogram of horizontal differences, see Figure 5.2.

Algorithm 5 Traverse one segment

Input: (α, s, L) associated data to segment, where α is initial angle of robot orientation at segment start, s is traveled distance and L is set of landmarks, a landmark is tuple (e, k, u, v, f, g), where e is a descriptor, k is counter of feature detections, u and v is position of feature in the image (at the moment of first, resp. last occurrence), f and g denotes distance from segment start according to u, resp. v.

Output: c steering gain parameter

$turn(\alpha)$
$d \leftarrow$ current traveled distance from the start.
while $d < s$ **do**
  $T \leftarrow \emptyset$
  $H \leftarrow \emptyset$
  $d \leftarrow$ current traveled distance from the start.
  $S \leftarrow$ extracted features, $(u, e) \in S$, u position, e feature descriptor.
  for all $\boldsymbol{l_i} = (e_i, k_i, u_i, v_i, f_i, g_i) \in L$ **do**
    $(u_a, e_a) \leftarrow \text{argmin}\{||(e_i, e(s))||, s \in S\}$
    if $g_i \geq d \geq f_i$ **then**
      $T \leftarrow T \cup \{\boldsymbol{t_i}\}$
    end if
  end for
  while $|T| > 0$ **do**
    $(e_i, k_i, u_i, v_i, f_i, g_i) \leftarrow \text{argmax}_{\boldsymbol{t} \in T}\, k(\boldsymbol{t})$
    $p \leftarrow \frac{d - f_i}{g_i - f_i}(v_i - u_i) + u_i$
    $(u_b, e_b) \leftarrow \text{argmin}\{||e_i, e(s)|| \; | s \in S \setminus \{(u_a, e_a)\}\}$
    if $||(e_i, e_a)|| < 0.8 \times ||(e_i, e_b)||$ **then** $H \leftarrow H \cup \{p_x - u_{ax}\}$
    end if
    $T \leftarrow T \setminus \{(e_i, k_i, u_i, v_i, f_i, g_i)\}$
  end while
  $\omega \leftarrow c \times \text{mode}(H)$
  $setSteering(\omega)$
end while

Figure 5.2: Robot view during autonomous navigation.

5.3.5 Histogram voting for steering correction

A key computation in the navigation algorithm is the estimation of the mode of the difference between the horizontal image coordinates of the mapped and recognized features. Since the differences are computed with a subpixel precision, we use a histogram voting method to find the mode. The histogram H has 31 bins, each bin represents an interval of 20 pixels. At the start of the autonomous navigation procedure 5, all bins of H are set to zero. Each time a difference is computed, the value of the corresponding bin is increased. Once the histogram is built, the mean value of all differences closer than 30 pixels to the center of the highest bin is computed and is considered to be the mode. Note that the histogram is built iteratively and that the mode computation is possible even when the main loop of the algorithm 5 is interrupted before all of the landmarks in T are processed. This property is important when real time constraints need to be satisfied.

Moreover, the histogram does not contain only the information about the mode, but also the congruence level of the observed and mapped landmarks. Simply put, in cases where the histogram maximum is clearly distinguishable, the computed steering value ω is likely to be correct. On the contrary, a flat histogram indicates a high number of incorrect correspondences and a danger that the steering value ω is wrong.

5.4 Stability

In this section, an insight into the stability of the method is shown. First, let's define a set of assumptions this method is based on:

- the robot moves in a plane,
- the map already exists in a form of a sequence of conjoined linear segments with landmark descriptions,
- the robot is able to recognize and associate a nonempty subset of mapped landmarks and to determine their bearing,
- the robot can (imprecisely) measure the traveled distance using odometry,
- the camera has a limited field of view and is aimed at the direction of the robot movement.

Let the path P be a sequence of linear segments p_i. The robot moves on a plane, i.e. its state vector is (x, y, φ). The robot we consider has a differential, nonholonomic drive and therefore $\dot{x} = \cos(\varphi)$ and $\dot{y} = \sin(\varphi)$. For each segment p_i, there exists a nonempty subset of landmarks for its traversal and a mapping between the robot position within the segment and expected bearing of each landmark is established. At the start of each segment, the robot resets its odometry counter and turns approximately towards the segment end to sense at least one of the segment landmarks. It establishes correspondences of seen and mapped landmarks and computes differences in expected and recognized landmark bearings. The robot steers in a direction that reduces those differences while moving forward until its odometry indicates that the current segment has been traversed.

Definition 1 (closed path navigation property). *Assume a robot navigates a closed path several times using an environment map only for heading corrections, while measuring the distance by odometry. The path for which the robot position uncertainty at any point is bound has a closed path navigation property.*

Theorem 1. *A path consisting of several conjoined non collinear segments retains the closed path navigation property if the conditions listed above are satisfied.*

Suppose that the given path is a square and the robot has to traverse it repeatedly. The robot is placed at a random (2D Gaussian distribution with zero mean) position near the first segment start, see Figure 5.3. The initial

position uncertainty can therefore be displayed as a circle, in which the robot is found with some probability. The navigation method is switched on and the robot starts to move along the first segment. Because it senses landmarks along the segment and corrects its heading, its lateral position deviation is decreased. However, due to the odometric error, the longitudinal position error increases. At the end of the segment, the circle denoting position uncertainty therefore becomes an ellipse with the shorter axis perpendicular to the segment. The effect of heading corrections is dependent on the lateral deviation, the greater the deviation, the stronger the effect of heading corrections and therefore the lateral error decreases by a factor h for every traversed segment. The odometry error is independent of the current position deviation and is influenced only by the length of the traversed segment and odometry multiplicative error. In our case, each segment has the same length and we can model the odometry error by an additive constant o.

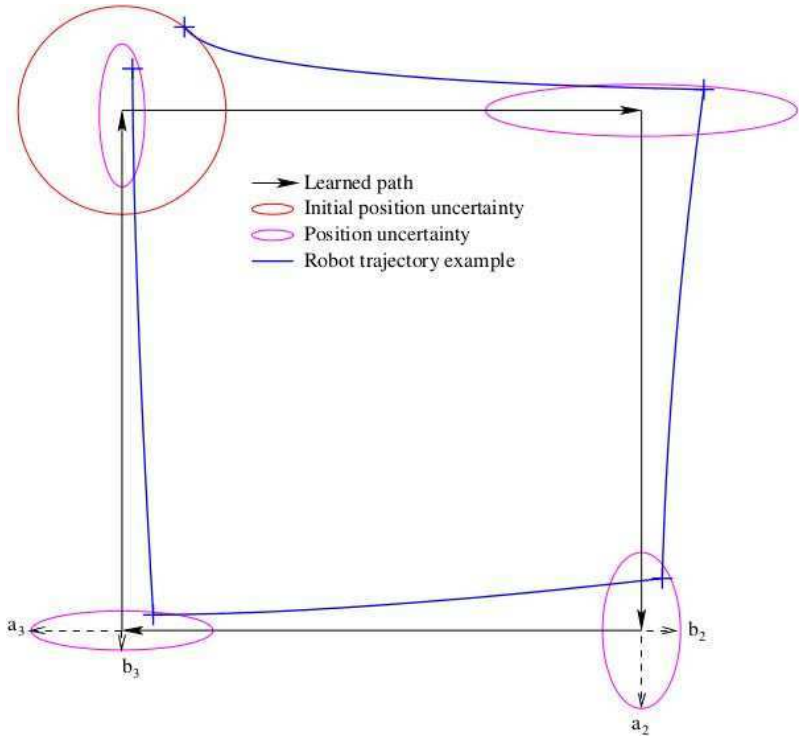


Figure 5.3: Position uncertainty evolution in a simple symmetric case.

After the segment is traversed, the robot turns 90° and starts to move along the second segment. The uncertainty changes again, but because of the direction change, the longer ellipse axis shrinks and the shorter one is elongated due to odometry error. As this repeats for every traversed segment, the size of the uncertainty ellipse might converge to a finite value. Since this particular trajectory is symmetric, we can easily compute each dimension of the ellipse and see if these values stabilize. We depict the longer and shorter

semiaxis of the i^{th} ellipse as a_i and b_i respectively, and establish the equations

$$\begin{aligned} a_{i+1} &= b_i + o \\ b_{i+1} &= a_i h, \end{aligned} \tag{5.1}$$

where h is a coefficient of lateral error reduction and o is the odometric error ($o = 1$, $h = 0.25$ in Figure 5.3). The equation (5.1) represents the influence of heading correction, which reduces the lateral uncertainty by the factor h and odometry measurement, which increases the longitudal error additively by o. Rewriting equations 5.1, we get

$$\begin{aligned} a_{i+2} &= a_i h + o \\ b_{i+2} &= h(b_i + o). \end{aligned} \tag{5.2}$$

If the factor of lateral position uncertainty decrease h is lower than one, the a_∞=$\lim_{i\to\infty} a_i$ and b_∞=$\lim_{i\to\infty} b_i$ exist and are obtainable by

$$\begin{aligned} a_\infty &= o/(1-h) \\ b_\infty &= ho/(1-h). \end{aligned} \tag{5.3}$$

This means that, as the robot travels the aforementioned path repeatedly, the dimensions of the ellipse characterizing its position uncertainty converge to a_∞ and b_∞. Note, that the a_∞ and b_∞ do not depend on the initial position uncertainty and existence of a finite solution of equation (5.2) does not depend on the particular value of the odometry error o. Though useful, this particular symmetric case gives us only a basic insight into the problem. A formal mathematical model of the navigation method for other nonsymmetrical paths as well and proof of convergence can be found in [49].

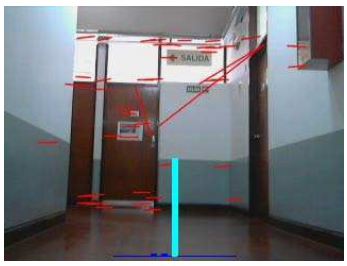
5.5 Experiments

To test the presented method, both indoor and outdoor experiments were performed. Also a convergence experiment was done to show the stability of the method described in 5.4

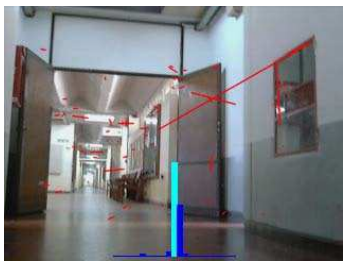
5.5.1 Indoor experiment

For the indoor experiment we performed a tour on the second floor of the Pabellón 1 building of Ciudad Universitaria, Buenos Aires. A path with a length of 32m, comprising three corridors and a hall, was first mapped in a guided way. After the path was mapped, the robot was placed in the same starting point of the learning phase and was requested to traverse it autonomously. It

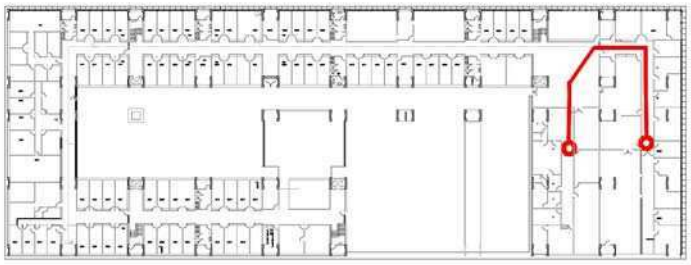
took approximately five minutes for the robot to traverse the learned path. The final position of the replay phase was 0.45m away from the final position of the learning phase, which represents around 1.4% error. Finally, the autonomous navigation was conducted again but stating 0.3 m away from the beginning of the path to the right. In this case, the final position of the replay phase was 0.93m away from final position of the learning phase, which represents around 2.9% error. In Figure 5.4 the path of the robot in the building and some snapshots of the autonomous navigation phase can be seen.



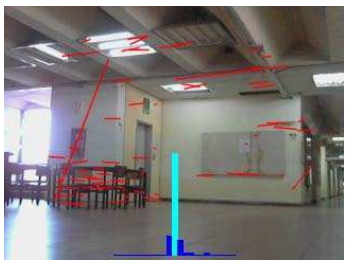
(a)



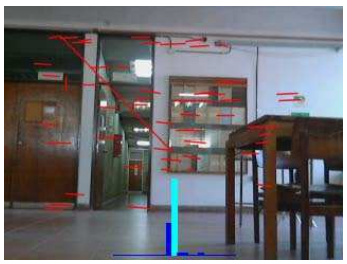
(b)



(c) 2nd floor plan of Pabellón 1, Ciudad Universitaria.



(d)



(e)

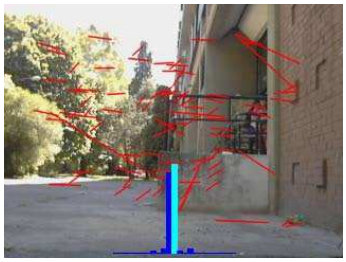
Figure 5.4: 5.4(a), 5.4(b), 5.4(d) and 5.4(e): snapshots (one for each path segment) extracted from autonomous navigation phase. 5.4(c): the path around offices corridors and hall in the 2nd floor of Pabellón 1, Ciudad Universitaria.

5.5.2 Oudoor experiment

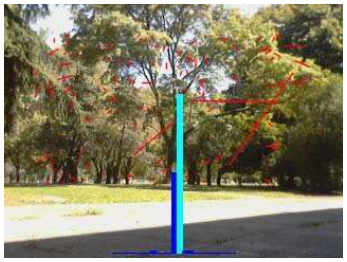
The method's performance for outdoor environments was evaluated in the university campus. The experiment was performed at the rear entrance of

the Pabellón 1 building. The robot was taught a 54m long path which went through a variable non-flat terrain with asphalt road and open regions. The approximate path is drawn on the map of the Pabellón 1 (see Figure 5.5).

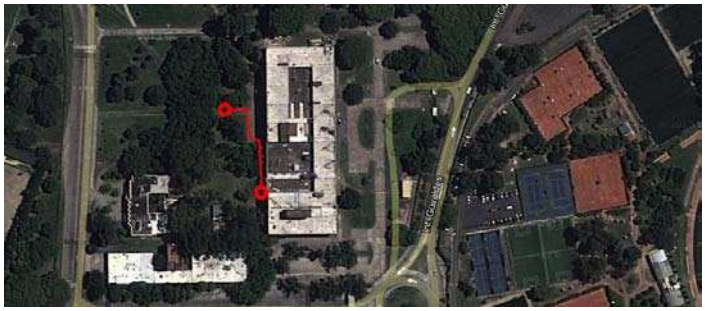
After the path had been mapped, the robot was placed at the beginning of the path and requested to traverse it autonomously. It took approximately teen minutes for the robot to traverse the learned path. Some pedestrians showed up and either entered the robot's field of view or crossed its path. Nevertheless, the robot was able to complete the learned path. The robot traversed the path autonomously with an accuracy of 0.57 m, which represents around 1.05% error.



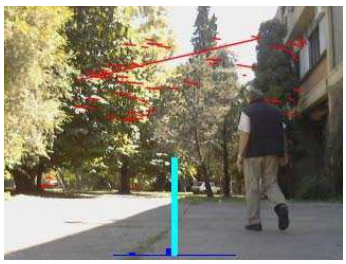
(a)



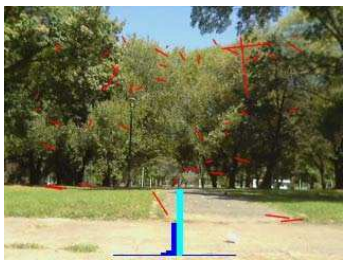
(b)



(c) Satellite image of Pabellón 1, Ciudad Universitaria.



(d)



(e)

Figure 5.5: 5.5(a), 5.5(b), 5.5(d) and 5.5(e): snapshots (one for each path segment) extracted from autonomous navigation phase. 5.5(c): the path around the rear entrance of the Pabellón 1 building.

5.5.3 Convergence experiment

The convergence experiment consists of doing the symmetrical case to test the stability of the method in practice (see Figure 5.3). The robot was guided around a square with a 5m side in a indoor environment. After the square path was mapped, the robot was placed 1m to the right of the starting spot and requested to traverse it autonomously. After three iterations the robot reached a place less than 0.05m from the starting spot. Figure 5.6 shows the three iteration final spots. The origin of coordinates coincides with the starting point of the learning phase.

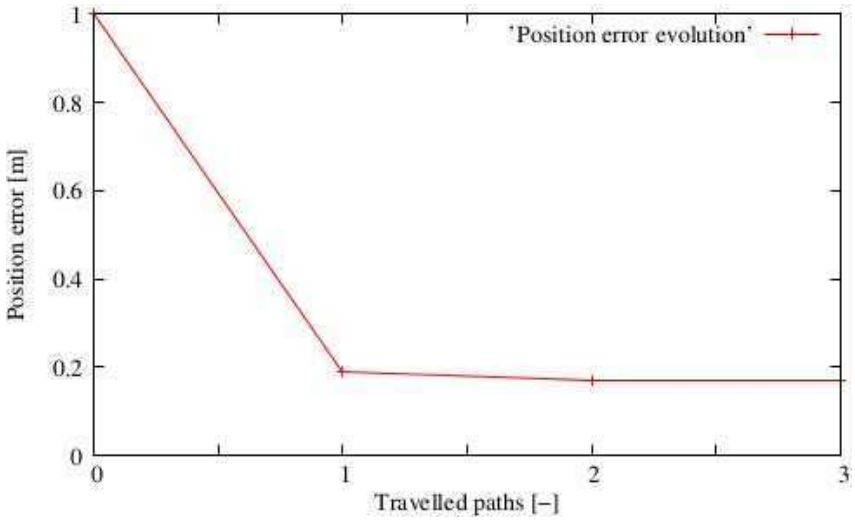


Figure 5.6: Position error evolution in a simple symmetric case.

5.6 Conclusions

In this Chapter, a landmark-based monocular navigation method and its implementation for the ExaBot were described. This method was based on the proposal of Tomas Krajnik [49]. It uses the map-and-replay technique. A topological map is built during learning phase which contains visual natural landmarks information about the environment. This landmarks are then used during the autonomous navigation phase. The basic idea is to utilize image features as visual landmarks to correct the robot's heading, leaving relative distance measurements to the odometry. The heading correction itself can suppress the odometric error and prevent the overall position error from diverging. The original version of the method uses a digital compass that was demonstrated to be unnecessary, since the turns of the robot can be estimated by odometry. Moreover, and more importantly instead of using SURF (Speeded Up Robust Features) as visual landmarks, we propose to use STAR based on

CenSurE (Center Surround Extremas) to detect image features and BRIEF (Binary Robust Independent Elementary Features) to describe them. In this way a significant improvement to the known method was proposed. Experiments with ExaBot were performed to test this method in both indoor and outdoor environment. Finally, a convergence experiment was also conducted to see the stability of the method in practice.

Although this method has shown to be robust and applicable to indoor/outdoor environments, it has some drawbacks. The main disadvantage is that the robot workspace is limited to only the regions visited during the training step. The user has to guide the robot all around the entire environment before performing autonomous navigation, which may represent a very tedious process, specially in large environments. In Chapter 7 we will propose a hybrid approach to take advantage of the segmentation-based navigation in the regions of the environment that can be traversed by the robot with a reactive control, i.e. without map. In this way, the only regions that need to be learned during the operator guide are those that can not be traversed without a map.

Chapter 6

Performace of local image features for long-term visual navigation

In this Chapter we evaluate a variety of image features for long-term visual navigation. The goal of this Chapter is to conclude which image feature detector-descriptor scheme is best for visual navigation.

6.1 Introduction

An open problem in mobile robotics is long-term autonomy in naturally changing environments. In this Chapter, we consider a scenario, in which a mobile robot running a map-and-replay method like the one described in Chapter 5 has to navigate a certain path over an extended -i.e. months- period of time. In long term, the perceived scene is not affected only by lighting conditions, but also by naturally occurring seasonal variations. A critical question is which kind of image feature detector-descriptor scheme would be the most robust to changes caused by lighting and seasonal variations. To address this question, we have used a dataset of sixty images covering an entire year of seasonal variations of an urban park to evaluate efficiency of state of the art image feature extractors.

Since we have shown that a crucial factor for mobile robot navigation is robot heading, we focus on estimation of the robot relative rotation among a set of images at a particular location. Therefore, we calculate relative rotation of each image pair from the same location and compare the results with a ground truth. A secondary measure of the feature extractor is its speed. To establish the speed, we have used the average time to extract a given number of features.

We start this Chapter with an overview of existing feature detector-descriptor methods. After that, we describe the dataset we have used for benchmarking and we propose two methods for image registration. Then we discuss results and reach a conclusion on about which type of the feature detector-descriptor scheme is best for long-term mobile robot navigation.

6.2 SIFT (Scale Invariant Feature Transform)

Scale-invariant feature transform (or SIFT) is the best known algorithm in computer vision to detect and describe local features in images. The algorithm was published by David Lowe in 1999 [154].

6.2.1 Keypoint detection

The first stage of keypoint detection is to identify locations and scales that can be repeatably assigned under differing views of the same object. Detecting locations that are invariant to scale changes of the image can be accomplished by searching for stable features across all possible scales, using a continuous function of scales known as scale space.

It has been shown by Lindeberg [155] that under some rather reasonable assumptions the only possible scale-space kernel is the Gaussian function. Therefore, the scale space of an image is defined as a function $L(x, y, \sigma)$ that is the convolution of a variable-scale Gaussian function $G(x, y, \sigma)$ with an input image $I(x, y)$

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{6.1}$$

where $*$ is the convolution operation in x and y, and

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2}\, e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{6.2}$$

To detect stable keypoint locations in scale space, Lowe has proposed to use scale-space extrema in the difference-of-Gaussian (DoG) function convolved with the image, $D(x, y, \sigma)$, which can be computed from the difference of two scales up to a constant multiplicative factor k:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) && (6.3)\\ &= L(x, y, k\sigma) - L(x, y, \sigma) && (6.4) \end{aligned}$$

For scale-space extrema detection in the SIFT algorithm, the image is first convolved with a Gaussian function at different scales. The convolved images are grouped by octaves. An octave correspondes to doubling the value of σ. The factors k_i are selected to obtain a fixed number of convolved images per octave. Then the difference-of-Gaussians (DoG) images are taken from Gaussian convolved images at adjacent scales in each octave. In Figure 6.1 the schema of how to compute DoG images is shown.

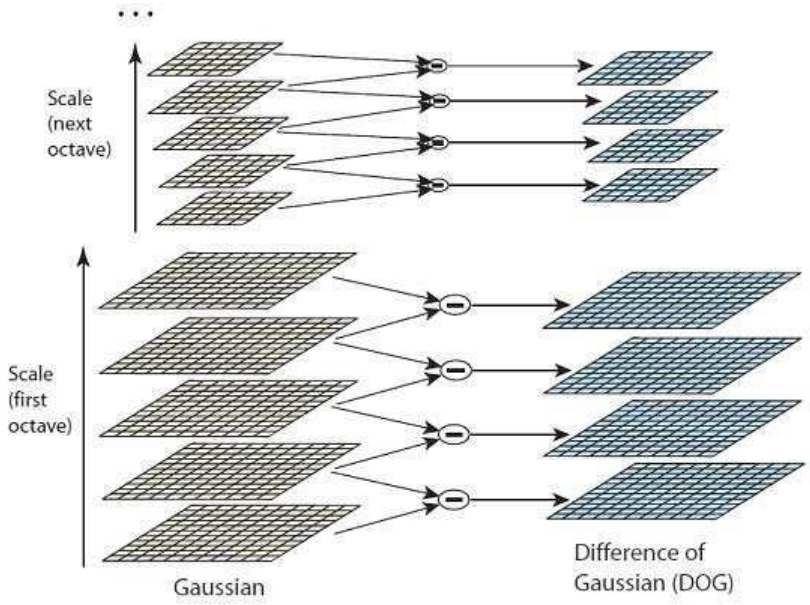


Figure 6.1: DoG images, extracted from [156]. For each octave of the scale space, the initial image is repeatedly convolved with Gaussians to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process is repeated.

In order to detect the local maxima and minima of $D(x, y, \sigma)$, each pixel is compared to its eight neighbors in the current DoG image and nine neighbors in the scale above and below. If the pixel value is the maximum or minimum among all compared pixels, it is selected as a candidate keypoint. This scale-space extrema detection produces too many keypoint candidates, some of which are unstable. The next step is to perform a detailed fit to the nearby data for accurate location, scale and ratio of principal curvature. This information allows points that have low contrast (and are therefore sensitive to noise) or are poorly localized along an edge to be rejected. To do that, the interpolated location of the extremum is calculated using the quadratic Taylor expansion of the DoG scale-space function $D(x, y, \sigma)$, with the candidate keypoint as the origin. This Taylor expansion is given by:

$$D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x} \tag{6.5}$$

where D and its derivatives are evaluated at the candidate keypoint $\mathbf{x_0}$ and $\mathbf{x} = (x, y, \sigma)$ is the offset from this point.

The location of the extremum $\hat{\mathbf{x}}$ is determined by taking the derivative of this function with respect to $\mathbf{x}$ and setting it to zero. If the offset $\hat{\mathbf{x}}$ is larger than 0.5 in any dimension, it is an indication that the extremun lies closer to another candidate keypoint. In this case, the candidate keypoint is changed and the interpolation is performed about that point instead. The final offset $\hat{\mathbf{x}}$ is added to its candidate keypoint to get the interpolated estimate for the location of the extremum.

To discard the keypoints with low contrast, the value of the second-order Taylor expansion $D(\mathbf{x})$ is computed at the final $\hat{\mathbf{x}}$. If this value is less than 0.03 the candidate keypoint is discarded. Otherwise it is kept, with final location $\mathbf{x_0} + \mathbf{x}$, where $\mathbf{x_0}$ is the original location of the candidate keypoint, and the scale σ.

The DoG function will have strong responses along edges, even if the candidate keypoint is not robust to small amounts of noise. Therefore, in order to increase stability, we need to eliminate the keypoints that have poorly determined locations but have high edge responses. For poorly defined peaks in the DoG function, the principal curvature across the edge would be much larger than the principal curvature along it. The principal curvatures can be computed from a 2×2 Hessian matrix, H, calcutaled at the location and scale of the keypoint:

$$H = \begin{pmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{pmatrix}$$

where D_{xx} denotes second partial derivative of DoG function $\frac{\partial^2 D}{\partial x^2}$ (and the same for D_{yy} and D_{xy}).

The eigenvalues of H are proportional to the principal curvatures of D. It turns out that the ratio of the two eigenvalues, if α is the larger one, and β the smaller one, is $r = \alpha/\beta$. The trace of H, given by $D_{xx} + D_{yy}$ is equal to the sum of the two eigenvalues, while its determinant, given by $D_{xx}D_{yy} - D_{xy}^2$ is equal to their product. Thus, the ratio of the trace squared and the determinant can be easily shown to be equal to $(r + 1)^2/r$, which depends only on the ratio of the eigenvalues rather than their individual values. This quantity is at a minimum when the eigenvalues are equal to each other and it increases with r. Therefore the higher the absolute difference between the two eigenvalues, which is equivalent to a higher absolute difference between the two principal curvatures of D, the higher the value of the ratio of the trace

and the determinant. Then, to check if the ratio of principal curvatures of a given keypoint is below some threshold, r, we only need to check if:

$$\frac{trace(H)^2}{det(H)} < \frac{(r+1)^2}{r}$$

When this condition is satisfied, it means that the keypoint is poorly localized and hence rejected. The experiments performed by Lowe use a value of $r = 10$, which eliminates keypoints that have a ratio between the principal curvatures greater than 10.

6.2.2 Keypoint orientation

Each keypoint is assigned one or more orientations based on local image gradient directions. This is the key step in achieving invariance to rotation as the descriptor can be represented relative to this orientation. First, the scale of the keypoint is used to select the Gaussian convolved image, with the closest scale, so that all computations are performed in a scale-invariant manner. For each image sample $L(x, y)$, at this scale, the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are precomputed using pixel differences:

$$m(x,y) = \sqrt{((L(x+1,y) - L(x-1,y))^2 + ((L(x,y+1) - L(x,y-1))^2}$$
$$\theta(x,y) = tan^{-1}(L(x,y+1) - L(x,y-1)/L(x+1,y) - L(x-1,y))$$

Note that m and θ do not depend on σ because this parameter is fixed to the closest scale of the keypoint. The magnitude and direction calculations for the gradient are done for every pixel in a neighboring region around the keypoint in the Gaussian convolved image L. An orientation histogram with 36 bins is formed, with each bin covering 10 degrees. Each sample in the neighboring window added to a histogram bin is weighted by its gradient magnitude and by a Gaussian-weighted circular window with a new σ that is 1.5 times that of the scale of the keypoint. The peaks in this histogram correspond to dominant orientations. Once the histogram is filled, the orientations corresponding to the highest peak and local peaks that are within 80% of the highest peaks are assigned to the keypoint. In the case of multiple orientations being assigned, an additional keypoint is created having the same location and scale as the original keypoint for each additional orientation.

6.2.3 Keypoint descriptor

Previous steps found keypoint locations at particular scales and assigned orientations to them. This ensured invariance to image location, scale and rotation.

Now we want to compute a descriptor vector for each keypoint such that the descriptor is highly distinctive and partially invariant to the remaining variations such as illumination, viewpoints, etc. This step is performed on the Gaussian convolved image closest in scale to the keypoint's scale.

To build the descriptor a patch around the keypoint is considered. In order to achieve orientation invariance, first the coordinates of this patch and the gradient orientations inside it are rotated relative to the keypoint orientation. Then, a set of orientation histograms are created on 4×4 subregions of the path. Each subregion has a 16×16 sample matrix and each histogram has 8 bins for directions. These histograms are computed from magnitude and orientation values of the samples. The magnitudes are further weighted by a Gaussian function with σ equal to one half the width of the descriptor window. These weighted magnitudes are then accumulated into the 8 bins of the corresponding histogram. The descriptor then becomes a vector of all the values of these histograms. Since there are $4 \times 4 = 16$ histograms each with 8 bins the vector has 128 elements (see Figure 6.2).

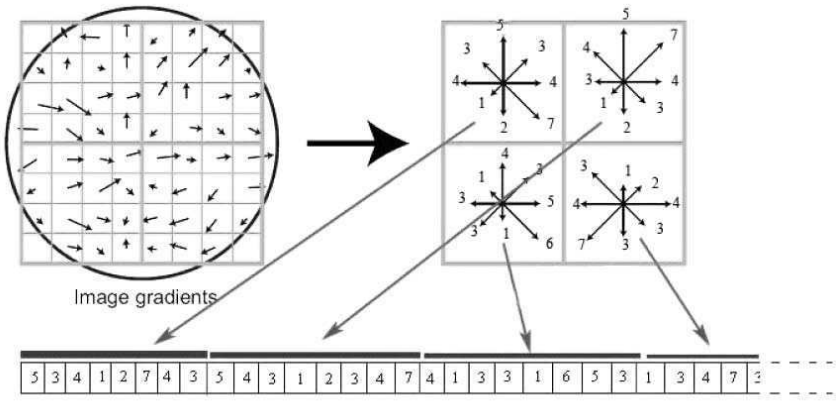


Figure 6.2: SIFT descriptor, extracted from [156]. It is built using $4 \times 4 = 16$ histograms and each histogram has 8 bins, then the descriptor vector has 128 elements.

Finally, the feature vector is modified to reduce the effects of illumination change. First, the vector is normalized. A change in image contrast in which each pixel value is multiplied by a constant will multiply gradients by the same constant, so this contrast change will be canceled by vector normalization. A brightness change in which a constant is added to each image pixel will not affect the gradient values, as they are computed from pixel differences. Therefore, the resulting descriptor is invariant to affine changes in illumination. However, non-linear illumination changes can also occur due to camera saturation or due to illumination changes that affect 3D surfaces with differing orientations by different amounts. These effects can cause a large change in

relative magnitudes for some gradients, but are less likely to affect the gradient orientations. The influence of large gradient magnitudes is reduced by thresholding the values in the unit feature vector using a 0.2 threshold, and then renormalizing the vector. This means that matching the magnitudes for large gradients is no longer as important, and that the distribution of orientations has greater emphasis. The value of 0.2 was determined experimentally using images containing different illuminations for the same 3D objects.

6.3 SURF (Speeded Up Robust Feature)

SURF (Speeded Up Robust Features) is a robust local feature detector and descriptor, first presented by Herbert Bay in 2006 [157]. It is inspired by the SIFT descriptor. The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT. One contribution of SURF algorithm is the use of integral images, which allows to speed-up the image processing.

6.3.1 Integral images

Integral images are used by all parts of SURF method to significantly accelerate their speed. The intregal image I_Σ is calculated from a luminance component of the original image I using the equation:

$$I_\Sigma(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j)$$

Using the integral image, only four memory read accesses are necessary to calculate the square integral of the original image, regardless of the area size. For example, the integral over the gray area S in Figure 6.3 is equal to $\Sigma(S) = I_\Sigma(A) + I_\Sigma(D) - I_\Sigma(C) - I_\Sigma(B)$.

6.3.2 Keypoint detection

SURF bases the keypoint detection on the Hessian matrix because of its good performance in computation speed and accuracy. The Hessian matrix can be used to detect blob-like structures at locations where its determinant is extremun. The value of the determinant is used to classify the maxima and minima of the function by the second order derivative test. Since the determinant is the product of eigenvalues of the Hessian, the locations can be classified

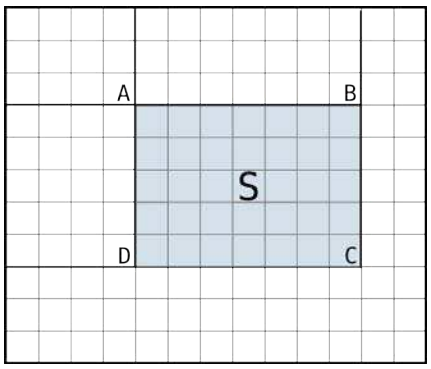


Figure 6.3: Integral image: it takes only four operations to calculate the gray area S of a rectangular region of any size.

based on the sign of the result. If the determinant is negative then the eigenvalues have different signs and hence the location is not a local extremum; if it is positive then either both eigenvalues are positive or both are negative and in either case the location is classified as an extremum.

Hence a method to calculate the second order partial derivatives of the image is required. This can be done by convolution with an appropriate kernel. In the case of SURF the second order scale normalized Gaussian is the chosen filter as it allows for analysis over scales as well as space. Kernels for the Gaussian derivatives in x, y and combined xy direction can be constructed such that we calculate the four entries of the Hessian matrix. The use of the Gaussian derivaties allows to vary the amount of smoothing during the convolution stage so that the determinant is calculated at different scales. Furthermore, since the Gaussian is an isotropic (i.e. circularly symmetric) function, convolution with the kernel allows for rotation invariance. Therefore, the Hessian matrix H can be calculated as a function of both space and scale $\mathbf{x} = (x, y, \sigma)$:

$$H(\mathbf{x}, \sigma) = \begin{pmatrix} L_{xx}(\mathbf{x}) & L_{xy}(\mathbf{x}) \\ L_{xy}(\mathbf{x}) & L_{yy}(\mathbf{x}) \end{pmatrix} \tag{6.6}$$

Here $L_{xx}(\mathbf{x})$ refers to the convolution of the second order Gaussian derivative $\frac{\partial^2 G(\mathbf{x})}{\partial x^2}$ with the image I at the point $\mathbf{x}$ and similarly for L_{xx} and L_{xy}. These derivatives are known as Laplacian of Gaussians (LoG). In order to use the integral image for convolution, and given Lowe's success with LoG approximations, the authors of SURF proposed to approximate Gaussian derivatives by box filters with integer coefficients. This approximation combined with the use of an integral image represents the biggest performance optimization in the SURF algorithm. The approximation is illustrated in Figure 6.4. The upper line contains representations of the discretized and cropped convolution of the

second order derivatives of Gaussian kernels L_{xx}, L_{yy}, L_{xy}, respectively. The lower line is their representation by simple box filters F_{xx}, F_{yy}, F_{xy}. Black squares in the representation of F_{xx} and F_{yy} mean weighting of each image pixel by coefficient -2. Black squares in representation of F_{xy} mean weighting of each image pixel by -1. On the other hand, white squares mean weighting coefficient of 1 and gray 0.

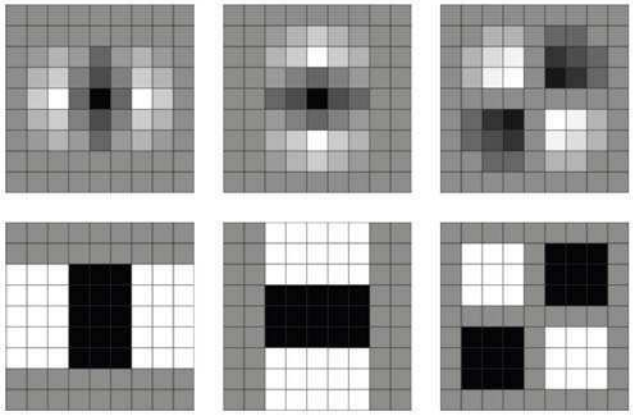


Figure 6.4: Top, left to right: discretized and cropped second order Gaussian derivatives in x (L_{xx}), y (L_{yy}) and xy-direction (L_{xy}), respectively. Down, left to right 2-D filter approximations F_{xx}, F_{yy} and F_{xy}. Extracted from [158].

The 9×9 pixel box filters are closest approximation of a Gaussian with $\sigma = 1.2$ and in the case of SURF represent the smallest scale to calculate the Hessian. This approximation presents a noticeable distortion to calculate the Hessian's determinant. Then, a relative weight w of the filter responses is used to balance the expression for the Hessian's determinant. This yields:

$$det(H) = (L_{xx}L_{yy}) - (L_{xy}L_{yx}) \tag{6.7}$$

$$det(H_{aprox}) = F_{xx}F_{yy} - (wF_{xy})^2 \tag{6.8}$$

It seeks to conserve energy between the Gaussian kernels and the approximated Gaussian kernels, then:

$$w = \frac{|L_{xy}(x, y, 1.2)|_F |F_{yy}(x, y, 9)|_F}{|L_{yy}(x, y, 1.2)|_F |F_{xy}(x, y, 9)|_F} = 0.912 \simeq 0.9 \tag{6.9}$$

where $|\cdot|_F$ is the Frobenius norm. Notice that for theoretical correctness, the weighting changes depending on the scale. In practice, w is constant, as this does not have a significant impact on the results.

6.3.3 Scale space construction

Because the SURF algorithm uses only simple box filters to build a detection response map, it is much more computationally efficient to scale box filters instead of the image (see Figure 6.5).

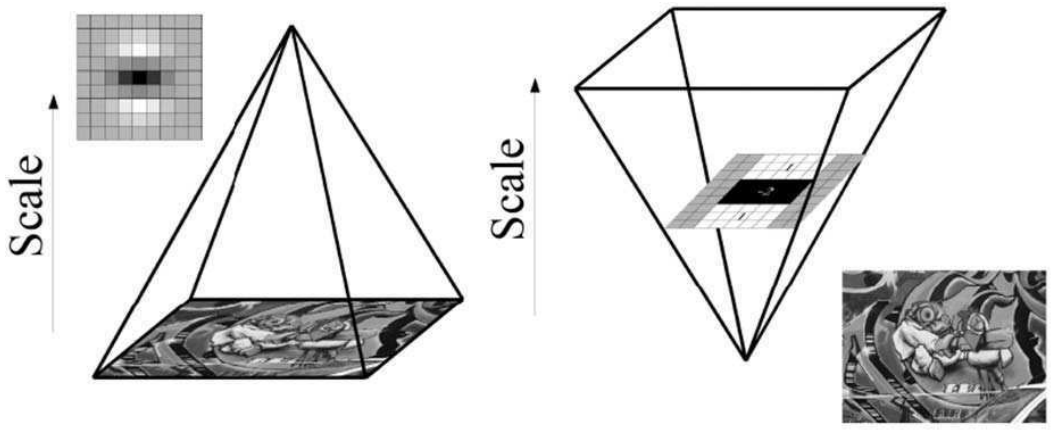


Figure 6.5: Scale space: instead of iteratively reducing the image size (left) as in SIFT, the use of integral images allows the up-scaling of the filter at constant cost in SURF (right). Extracted from [158].

Box filter scaling with the use of integral image keeps the calculation time constant for all filter sizes. This approach is mathematically equivalent to scaling in image pyramid and calculating the response map afterward. The smallest filter size is 9×9 pixels and, as mentioned before, corresponds to a Gaussian kernel with $\sigma = 1.2$. This level is denoted as a 'base scale'. To preserve symmetry, the box filter areas must increase evenly and symmetrically. This means that we have to add at least 6 pixels to a filter edge during each step in a scale space creation. As in SIFT, the scale space is divided into octaves. An octave represents a series of filter response maps obtained by convolution using increasing filter sizes. In total, an octave covers a scaling factor of 2 (which implies that the filter size has to be more than doubled). Each octave is subdivided into a constant number of scale levels (called intervals). To cover the scale space efficiently the sampling step and filter size increase step doubles between subsequent octaves. For two successive levels, we must increase the size of the filter by a minimum of 2 pixels (1 pixel on every side) in order to keep the size uneven and thus ensure the presence of the central pixel. This yields in a total increase of the mask size by 6 pixels (see Figure 6.6).

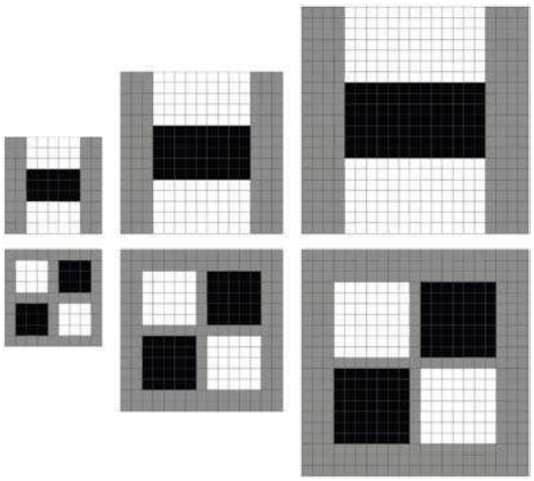


Figure 6.6: Filters F_{yy} (top) and F_{xy} (bottom) fo three successive scale levels: 9×9, 15×15 and 21×21. Extracted from [158].

6.3.4 Keypoint location

At this point the Hessian values in the desired octaves and intervals are calculated filling a scale space. In SURF, keypoint location is composed of three steps. Determinants are first compared with a required lower threshold value. This value controls the overall sensitivity of the detector. Determinants which surpass the threshold are subjected to local non-maximum suppression during which the candidate determinant is compared with all of its 26 neighbors in scale space, like in SIFT. Because the candidate determinant must surpass all of its neighbors, interest points can be located only in scale intervals with both neighbor scale levels.

The last step in the keypoint localization is a sub-pixel position interpolation of the Hessian function local maximum. The 2nd. order Taylor approximation of the Hessian function in scale space is used:

$$H(\mathbf{x}) = H + \frac{\partial H^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T\frac{\partial^2 H}{\partial \mathbf{x}^2}\mathbf{x} \tag{6.10}$$

As in SIFT, the location of the extremum $\hat{\mathbf{x}}$ is determined by taking the derivative of this function with respect to $\mathbf{x}$ and setting it to zero. If the resulting location of an interpolated local maximum has all its absolute values of coordinates smaller than 0.5, the candidate determinant position is fixed (after proper scaling) with interpolation result and accepted as a keypoint location. If some of the local maximum coordinates is bigger than 0.5, then the candidate determinant position is updated and the interpolation procedure is repeated. This procedure is applied until either the criterion is satisfied or the maximal number of interpolation steps is reached.

6.3.5 Keypoint orientation

The rotation invariance mechanism in SURF uses dominant luminance gradient orientation in the neighborhood of the keypoint. The keypoint circular neighborhood with radius $6s$ (with s the scale at which the keypoint was detected) is sampled using $4s$-sized Haar wavelets in s sized steps. Once the wavelet responses are calculated and weighted with a Gaussian function with $\sigma = 2s$ centered at the keypoint, they are represented as points in a 2-D space with the horizontal response strength along the abscissa and the vertical response strength along the ordinate. The vectors from the origin to all points in a sliding orientation window of size $\pi/3$ are summed to form a dominant orientation vector for a given window. The sliding orientation window is shifted in $\pi/18$ steps. The orientation of the biggest summed vector is selected as a reproducible keypoint orientation (see Figure 6.7.

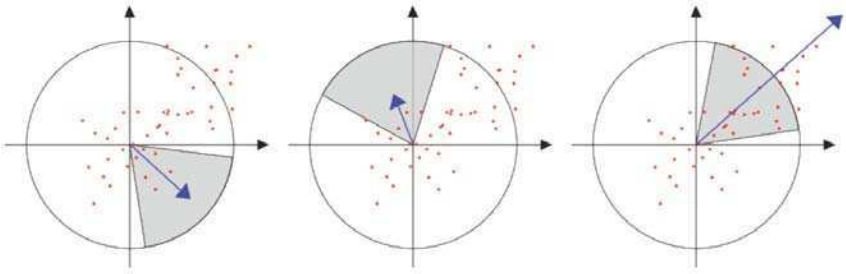


Figure 6.7: Orientation assignment: a sliding orientation window of size $\pi/3$ detects the dominant orientation of the Gaussian weighted Haar wavelet responses at every sample point within a circular neighborhood around the keypoint. Extracted from [158].

Note that for some applications, such as ground robot navigation in a flat terrain, rotation invariance is not necessary, thus they do not use this step. This alternative is called U-SURF (Upright version of SURF).

6.3.6 Keypoint descriptors

The descriptor is calculated from a square area surrounding the keypoint with edge size $20s$ (s the scale of the keypoint). If rotation invariance is required, the square area has to be rotated by the dominant orientation angle of the keypoint. This area is split up regularly into 4×4 square sub-regions. These sub-regions are regularly sampled using Haar wavelet responses with sampling step s, producing $5 \times 5 = 25$ samples per sub-region. If the rotation invariance is required, the responses should be calculated from a rotated image, however due to the usage of integral images to achieve higher computational efficiency, the responses are calculated from the original image and afterward rotated by

the dominant orientation angle. Wavelet responses in the Figure 6.8 represent already rotated horizontal and vertical responses. The response rotated along the dominant orientation angle is denoted dx and the response perpendicular in a positive sense dy. To increase descriptor robustness against geometric deformations and localization errors, the wavelet responses are weighted by a Gaussian with $\sigma = 3.3s$ centered at the keypoint. Each of the 16 sub-regions is afterward described by a four-dimensional vector defined as $\mathbf{v} = (\sum dx, \sum dy, \sum |dy|, \sum |dx|)$. The resulting SURF descriptor is obtained by putting all $4 \times 4 \times 4 = 64$ values into one vector and normalizing it.

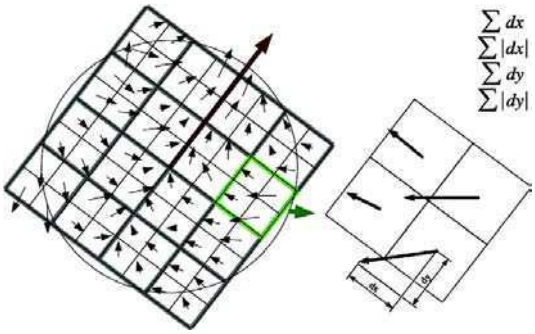


Figure 6.8: To build the descriptor, an oriented quadric grid with 4×4 square sub-region is laid over the keypoint (left). For each square, the wavlet responses are computed from 5×5 (for illustrative purposes, we show only 2×2 su-divisions). For each sub-division, dx, $|dx|$, dy and $|dy|$ are computed relatively to the orientation of the grid (right). Extracted from [158].

6.4 CenSurE (Center Surround Extrema)

Center Surround Extrema (or CenSurE) has been proposed by Motilal Agrawal et. al. in 2008 [159]. It is developed to be used in real time applications. The CenSurE features are computed at the extrema of the center-surround filters over multiple scales.

6.4.1 Bi-level Filters

Scale invariant detectors based on image pyramid are known to loose precision in the higher levels of the pyramid due to sub-sampling. SIFT, SURF and other algorithms which use sub-sampling try to recover this precision by calculating the keypoint location with sub-pixel interpolation. CenSurE detection method abandons this approach and searches for keypoints on all scales using the same precision. Keypoints are detected as extrema of the Laplacian in scale-space, or more generally, extrema of the center-surround response. While Lowe

approximated the Laplacian with the difference of Gaussians, CenSurE seeks even simpler approximations, using center-surround filters that are bi-level, that is, they multiply the image value by either 1 or -1. Figure 6.9 shows a progression of bi-level filters with varying degrees of symmetry. The circular filter is the most faithful to the Laplacian, but hardest to compute. The other filters can be computed rapidly with integral images, with decreasing cost from octagon to hexagon to box filter.

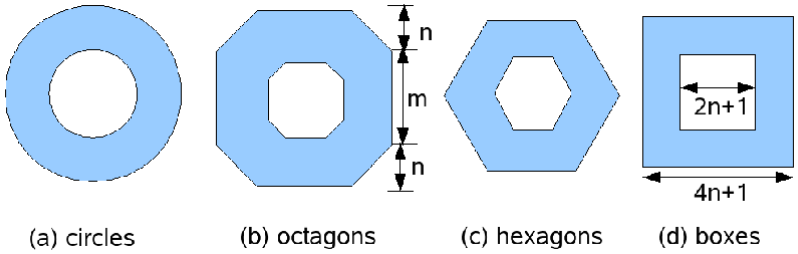


Figure 6.9: Progression of Center-Surround bi-level filters. (a) circular symmetric BLoG (Bilevel LoG) filter. Successive filters: (b) octagon (c) hexagon, (d) box have less symmetry. Extracted from [159]

The two endpoints are analyzed: octagons for good performance, and boxes for good computation.

CenSurE-DOB

The idea is to replace the two circles in the circular BLoG with squares to form the CenSurE-DOB (CenSurE difference of Boxes). This results in a basic center-surround Haar wavelet. Figure 6.9(d) shows the generic center-surround wavelet of block size n. The inner box is of size $(2n+1) \times (2n+1)$ and the outer box is of size $(4n+1) \times (4n+1)$ (see Figure 6.9(d)).

CenSurE-Oct

Difference of Boxes are obviously not rotationally invariant kernels. In particular, DOBs will perform poorly for 45 degrees in-plane rotation. Octagons, on the other hand are closer to circles and approximate LoG better than DOB. An octagon can be represented by the height of the vertical side (m) and height of the slanted side (n) (see Figure 6.9(b)).

STAR

STAR is derived from CenSurE detector. While CenSurE uses polygons, STAR mimics the circle with 2 overlapping squares: one upright and other 45-degree rotated. These polygons are also bi-level. They can be seen as polygons with thick borders. The borders and the enclosed area have weights of opposing signs. STAR is the most popular implementation of CenSurE detector schema.

Filter Computation

The key of CenSurE is the possibility to compute the bi-level filters efficiently at all sizes. The box filter can be implemented using the integral image. Modified versions of integral images can be exploited to compute the other polygonal filters rather than rectangular areas. The idea here is that any trapezoidal area can be computed in constant time using a combination of two different slanted integral images, where the sum at a pixel represents an angled area sum. The degree of slant is controlled by a parameter α:

$$I_{\Sigma_\alpha}(x,y) = \sum_{i=1}^{y} \sum_{j=1}^{x+\alpha(y-i)} I(i,j)$$

When $\alpha = 0$, this is just the standard rectangular integral image. For $\alpha < 0$, the summed area slants to the left; for $\alpha > 0$, it slants to the right. Slanted integral images can be computed in the same time as rectangular ones, using incremental techniques. Adding two areas together with the same slant determines one end of a trapezoid with parallel horizontal sides; the other end is done similarly, using a different slant. Each trapezoid requires three additions, just as in the rectangular case. Finally, the polygonal filters can be decomposed into 1 (box), 2 (hexagon), and 3 (octagon) trapezoids, which is the relative cost of computing these filters.

6.4.2 Keypoint detection

First, seven filter responses at each pixel in the image are computed. Then, a non-maximal suppression over the scale space is performed. Briefly, a response is suppressed if there is a response greater (maxima case) or a response less than (minima case) its neighbors, in a local neighborhood over the location and scales. Pixels that are either maxima or minima in this neighborhood are the keypoint locations. A $3 \times 3 \times 3$ neighborhood is used for non-maximal suppression. The magnitude of the filter response gives an indication of the strength of the feature. The greater the strength, the more likely it is to be repeatable. Weak responses are likely to be unstable. Therefore, a threshold

can be applied to filter out the weak responses. Since all responses are computed on the original image without sub-sampling, all keypoint are localized correctly so subpixel interpolation is not needed.

After searching for local extrema in scale space the line suppression mechanism is applied. It is based on the scale-adapted Harris measure computed from the box filter responses. The Harris measure is more expensive to compute than the Hessian matrix used by SIFT. However, this measure needs to be computed for only a small number of keypoints that are scale-space maxima and whose response is above a threshold and hence does not represent a computational bottleneck.

6.4.3 Keypoint descriptor

The CenSurE descriptor is an improvement of the SURF descriptor. The SURF descriptor weighs the Haar wavelet responses. This may cause boundary effects in which the descriptor abruptly changes, yielding poor results. To account for these boundary conditions, each boundary in CenSurE descriptor has a padding of $2s$, thereby increasing the region size for the descriptor from $20s$ to $24s$, being s the scale of the keypoint. The Haar wavelet responses in the horizontal (dx) and vertical (dy) directions are computed for each 24×24 point in the region with filter size $2s$ by first creating a summed image, where each pixel is the sum of a region of size s. The Haar wavelet output results in four fixed-size dx,dy, $|dx|$,$|dy|$ images that have the dimensions 24×24 pixels irrespective of the scale.

Each dx,dy, $|dx|$,$|dy|$ image is then split into 4×4 square overlapping subregions of size 9×9 pixels with an overlap of 2 pixels with each of the neighbors. Figure 6.10 shows these regions and subregions. For each subregion the values are then weighted with a precomputed Gaussian function (with $\sigma = 2.5$) centered on the subregion center and summed into the usual SURF descriptor vector for each subregion: $\mathbf{v} = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$. Each subregion vector is then weighted using another Gaussian function (with $\sigma = 1.5$) defined on a mask of size 4×4 and centered on the feature point. Like the original SURF descriptor, this vector is then normalized. The achieved overlap allows each subregion to work on a larger area so samples that get shifted around are more likely to still leave a signature in the correct subregion vectors. Likewise, the subregion Gaussian weighting means that samples near borders that get shifted out of a subregion have less impact on the subregion descriptor vector.

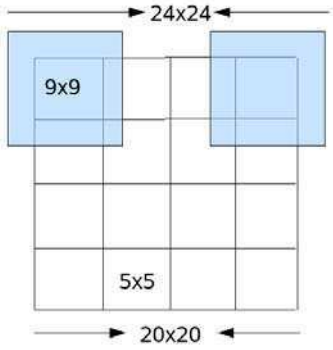


Figure 6.10: Regions and subregions for the CenSurE descriptor. Each subregion (in blue) is 9×9 with an overlap of 2 pixels at each boundary. All sizes are relative to the scale of the keypoint s.

6.5 BRIEF (Binary Robust Independent Elementary Features)

BRIEF (Binary Robust Independent Elementary Features) is a keypoint descriptor algorithm, presented by Michael Calonder in 2010 [160]. The key advantage of BRIEF is to use binary strings as an efficient feature point descriptor.

6.5.1 Keypoint descriptor

The BRIEF approach is based on the idea that image patches can be effectively classified by a relatively small number of pairwise intensity comparisons. To do this, a test τ on a patch $\mathbf{p}$ of size $S \times S$ is defined as:

$$\tau(\mathbf{p}, \mathbf{x}, \mathbf{y}) = \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases} \tag{6.11}$$

where $\mathbf{p}(\mathbf{x})$ is the pixel intensity in a smoothed version of $\mathbf{p}$ at location $\mathbf{x} = (u, v)^T$. Choosing a set of n $(\mathbf{x}, \mathbf{y})$-locations pairs uniquely defines a set of binary tests. Then, BRIEF descriptor is built as n-dimensional bitstring

$$f_n(\mathbf{p}) = \sum_{i=1}^{i=n} 2^{i-1} \tau(\mathbf{p}, \mathbf{x}_i, \mathbf{y}_i). \tag{6.12}$$

In [160], the authors consider $n = 128$, 256 and 512 referred as BRIEF-k where $k = n/8$ represents the number of bytes required to store the descriptor.

Generating a $n-$bit long vector leaves many options or strategies for selecting the n test locations $(\mathbf{x}_i, \mathbf{y}_i)$ of Equation 6.11 in a patch of size $S \times S$. The authors of BRIEF experimented with five sampling geometries depicted by Figure 6.11 Assuming the origin of the patch coordinate system to be located at the patch center, they can be described as follows:

1. $(\mathbf{X}, \mathbf{Y}) \sim$ i.i.d. (independent and identically distributed) Uniform $(-\frac{S}{2}, \frac{S}{2})$: The $(\mathbf{x}_i, \mathbf{y}_i)$ locations are evenly distributed over the patch tests can lie close to the path border.

2. $(\mathbf{X}, \mathbf{Y}) \sim$ i.i.d. Gaussian $(0, \frac{1}{25}S^2)$: The tests are sampled from an isotropic Gaussian distribution. Experimentally $\frac{S}{2} = \frac{5}{2}\sigma \leftrightarrow \sigma^2 = \frac{1}{25}S^2$ to give best results in terms of recognition rate.

3. $\mathbf{X} \sim$ i.i.d. Gaussian $(0, \frac{1}{25}S^2)$ and $\mathbf{Y} \sim$ i.i.d. Gaussian $(\mathbf{x}_i, \frac{1}{100}S^2)$. The first location $\mathbf{x}_i$ is sampled from a Gaussian distribution centered around the origin while the second location is sampled from another Gaussian centered on $\mathbf{x}_i$. This forces the tests to be more local. Test locations outside the patch are clamped to the edge of the patch. Again, $\sigma^2 = \frac{1}{100}S^2$ is setted experimentally.

4. $(\mathbf{x}_i, \mathbf{y}_i)$ are randomly sampled from discrete locations of a coarse polar grid introducing a spatial quantization.

5. $\mathbf{x}_i = (0, 0)^T$ and y_i takes all possible values on a coarse polar grid containing n points.

For each of these test geometries, recognition rates were computed and authors concluded that $(\mathbf{X}, \mathbf{Y}) \sim$ i.i.d. Gaussian $(0, \frac{1}{25}S^2)$ has the best results so this strategy is used for BRIEF.

The main advantage of describing the keypoints with strings of bits (called binary descriptors) is that its similarity can be measured by the Hamming distance. This distance can be computed extremely fast on modern CPUs that often provide a specific instruction to perform a XOR or bit count operation, as is the case in the latest SSE instruction set. This means that BRIEF easily outperforms other fast descriptors such as SIFT or SURF.

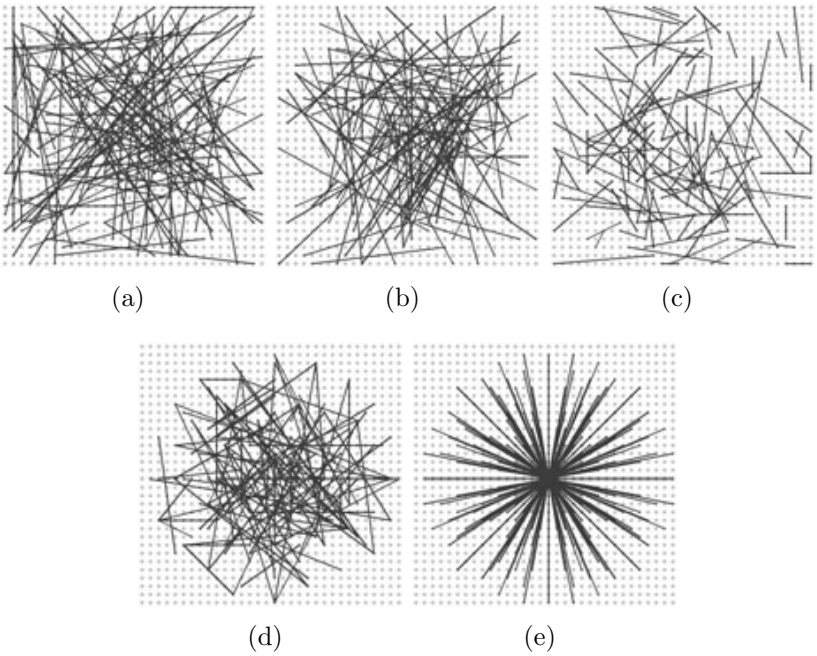


Figure 6.11: Different approaches to choosing the test locations. All except the righmost one are selected by random sampling. Showing 128 tests in every image. Extracted from [160].

6.6 BRISK (Binary Robust Invariant Scalable Keypoints)

BRISK (Binary Robust Invariant Scalable Keypoints) is another image features detector-descriptor schema, proposed by Stefan Leutenegger et. al. [161] in 2011. The authors claim that BRISK can be used for tasks with hard real-time constraints or limited computation power and, at the same time, reach precision similar to SIFT or SURF. The key to speed lies in the application of a novel scale-space FAST-based detector in combination with the assembly of a bit-string descriptor from intensity comparisons retrieved by dedicated sampling of each keypoint neighborhood.

6.6.1 Keypoint detection

With the focus on efficiency of computation, the BRISK detector is inspired by the AGAST detector [162]. In turn, AGAST is essentially an extension for accelerated performance of the now popular FAST detector [163], proven to be a very efficient basis for feature extraction. With the aim of achieving invariance to scale, which is crucial for high-quality keypoints, BRISK goes one step further by searching for maxima not only in the image plane, but

also in scale-space using the FAST score as a measure for saliency. Despite discretizing the scale axis at coarser intervals like SURF, the BRISK detector estimates the true scale of each keypoint in the continuous scale-space. A keypoint is identified at octave by analyzing the 8 neighboring saliency scores in that octave as well as in the corresponding scores-patches in the immediately-neighboring layers above and below. In all three layers of interest, the local saliency maximum is sub-pixel refined before a 1D parabola is fitted along the scale-axis to determine the true scale of the keypoint. The location of the keypoint is then also re-interpolated between the patch maxima closest to the determined scale (see Figure 6.12).

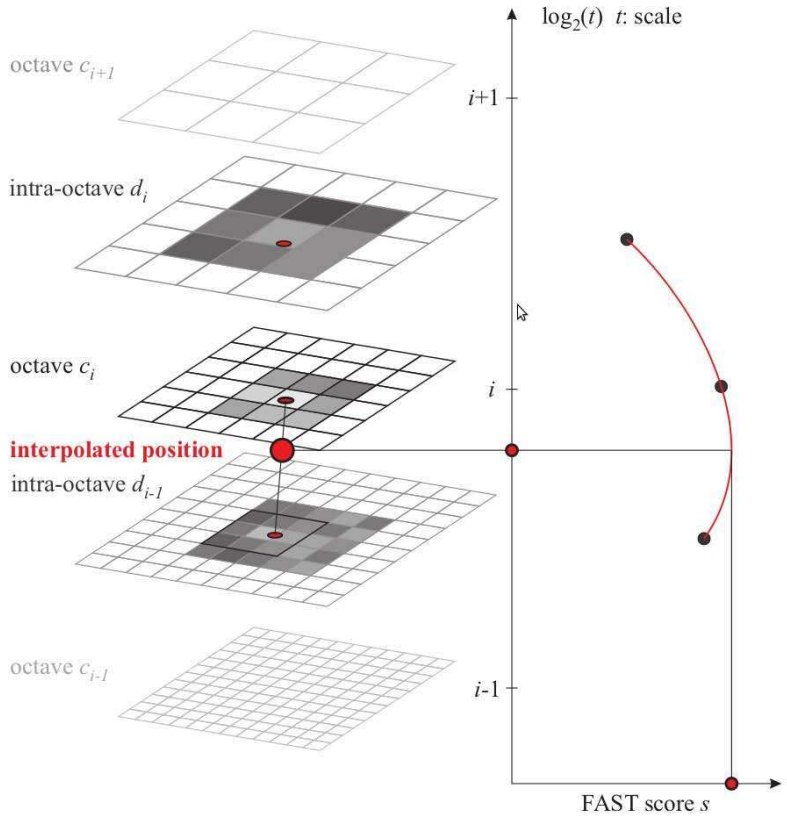


Figure 6.12: Scale-space interest point detection: in all three layers of interest, the local saliency maximum is sub-pixel refined before a 1D parabola is fitted along the scale-axis to determine the true scale of the keypoint. Extracted from [161].

6.6.2 Keypoint descriptor

Given a set of keypoints (consisting of sub-pixel refined image locations and associated floating-point scale values), the BRISK descriptor is composed as a binary string by concatenating the results of simple brightness comparison

tests. The idea is the same as in BRIEF, however here it is employed in a more qualitative manner. In BRISK, the characteristic direction of each keypoint is identified to allow for orientation-normalized descriptors and hence achieve rotation invariance which is key to general robustness. Also, the brightness comparisons are carefully selected with the focus on maximizing descriptiveness.

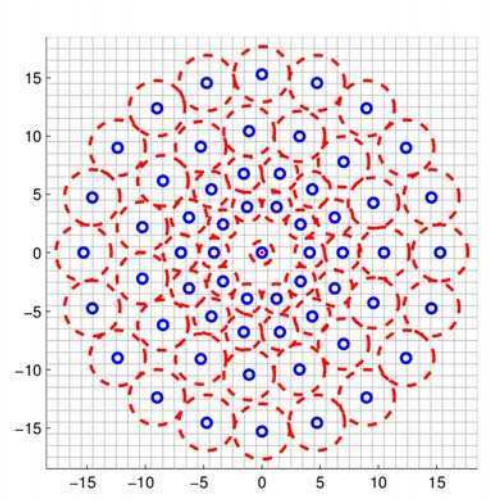


Figure 6.13: The BRISK sampling pattern with $N = 60$ points: the small blue circles denote the sampling locations; the bigger, red dashed circles are drawn at a radius σ corresponding to the standard deviation of the Gaussian kernel used to smooth the intensity values at the sampling points. Extracted from [161].

The BRISK descriptor uses pattern for sampling the neighborhood of the keypoint. The pattern, illustrated in Figure 6.13, defines N locations equally spaced on circles concentric with the keypoint. In order to avoid aliasing effects when sampling the image intensity of a point p_i in the pattern, a Gaussian smoothing function with standard deviation σ_i proportional to the distance between the points on the respective circle is applied. Positioning and scaling the pattern accordingly for a particular keypoint k in the image, let us consider one of the $N(N-1)/2$ sampling-point pairs (p_i, p_j). The smoothed intensity values at these points which are $I(p_i, \sigma_i)$ and $I(p_j, \sigma_j)$ respectively, are used to estimate the local gradient $g(p_i, p_j)$ by

$$g(p_i, p_j) = (p_j - p_i)\frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{\|p_j - p_i\|^2}.$$

Considering the set A of all sampling point pairs:

$$A = \{(p_i, p_j) \in \mathbb{R}^2 \times \mathbb{R}^2 \mid i < N \wedge j < i \wedge i, j \in \mathbb{N}\}$$

two subset are defined, one of short pairings S and another subset of L long-distance pairings:

$$S = \{(p_i, p_j) \in A \mid \|p_j - p_i\| < \delta_{max}\} \subseteq A$$
$$L = \{(p_i, p_j) \in A \mid \|p_j - p_i\| > \delta_{min}\} \subseteq A$$

where δ_{max} and δ_{min} are distances thresholds. Iterating through the point pairs in L, overall characteristic pattern direction of the keypoint k is estimated as:

$$g = (g_x, g_y)^\top = \frac{1}{L} \sum_{(p_i, p_j) \in L} g(p_i, p_j)$$

The long-distance pairs are used for this computation, based on the assumption that local gradients annihilate each other and are thus not necessary in the global gradient determination. For the formation of the rotation- and scale-normalized descriptor, BRISK applies the sampling pattern rotated by $\alpha = \arctan2(g_y, g_x)$ around the keypoint k. The bit-vector descriptor d_k is assembled by performing all the short distance intensity comparisons of point pairs $(p_i^\alpha, p_j^\alpha) \in S$ (i.e. in the rotated pattern), such that each bit b corresponds to:

$$b = \begin{cases} 1 & \text{if } I(p_j^\alpha, \sigma_j) > I(p_i^\alpha, \sigma_i) \\ 0 & \text{otherwise} \end{cases} \qquad \forall (p_i^\alpha, p_j^\alpha) \in S \tag{6.13}$$

Finally, matching two BRISK descriptors is a simple computation of their Hamming distance as done in BRIEF.

6.7 ORB

ORB is a detector-descriptor schema based on the FAST detector and the BRIEF descriptor. It has been presented by Ethan Rublee et. al. [164] in 2011. The main properties of ORB are rotation invariant and noise resistance.

6.7.1 Keypoint detection

First FAST-9 (circular radius of 9) detector is used for keypoints detection. As FAST does not produce a measure of cornerness, it has large responses along edges. Thus, Harris corner measure [165] is employed to sort the FAST

keypoints. In order to measure corner orientation, the intensity centroid [166] is used. The use of the intensity is based on the assumption that a corner's intensity is offset from its center, and this vector may be used to compute an orientation. Thus, the moments of a patch are defined as:

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y), \tag{6.14}$$

and with these moments, the centroid can be found:

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \tag{6.15}$$

Then, the vector from the corner center O to the centroid C can be defined and the orientation of the patch is simply:

$$\theta = \text{atan2}(m_{01}, m_{10}) \tag{6.16}$$

where atan2 is the quadrant-aware version of arctangent.

6.7.2 Keypoint descriptor

The ORB descriptor is based on BRIEF. To achieve rotational invariance the idea is to steer the BRIEF descriptor according to the orientation of the keypoint. For each keypoint, a set of n binary tests at location $(\mathbf{x}_i, \mathbf{y}_i)$ define a $2 \times n$ matrix:

$$S = \begin{pmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \\ \mathbf{y}_1 & \cdots & \mathbf{y}_n \end{pmatrix} \tag{6.17}$$

Using the patch orientation θ and the corresponding rotation matrix R_θ, a "steered" version S_θ of S can be constructed:

$$S_\theta = R_\theta S, \tag{6.18}$$

and then the steered BRIEF descriptor of equation (6.12) becomes:

$$g_n(\mathbf{p}, \theta) = f_n(\mathbf{p}) | (\mathbf{x}_i, \mathbf{y}_i) \in S_\theta \tag{6.19}$$

The angle is discretized into increments of $2\pi/30$ (12 degrees), and a lookup table of precomputed BRIEF patterns is constructed. As long at the keypoint orientation θ is consistent across views, the correct set of points S_θ will be used to compute its descriptor.

6.8 The dataset

The testing data set was created by Tomas Krajnik over the period of one year in a large outdoor environment, Stromovka park in Prague, Czech Republic, under variety of light conditions. Each month, the robot was driven by means of teleoperation through a 50 meter long path consisting of five straight segments while capturing images by its onboard camera. The dataset comprises images taken at the start of each segment, i.e. it contains 60 images from five distinct locations - we denote a picture from n^{th} month and m^{th} location as $\mathcal{I}_n^m$. Although the path started and ended on exactly the same place, a slight variation of the path has been introduced every time for testing purposes. Therefore, the first set of images $\mathcal{I}_1^1, \mathcal{I}_2^1 \ldots \mathcal{I}_{12}^1$ is taken from exactly the same location, while the locations of others vary about ± 1 m. Captured images contain several environment changes caused by seasonal factors, moving objects, weather etc. Figure 6.14 shows views from the robot camera at the second location for each month of the year.

(a) November (e) December (i) January

(b) February (f) March (j) April

(c) May (g) June (k) July

(d) August (h) September (l) October

Figure 6.14: Views from the robot camera at the second location taken once a month through one year.

The intrinsic parameters of the camera were found using the MATLAB calibration toolbox and radial distortion of the images was removed by the same tool [167]. The resulting images had their bottom half removed, because the

bottom part contains only ground, which is not relevant for image matching.

6.9 Feature evaluation

Since we have shown that a crucial factor for mobile robot navigation is the robot heading, we focus on the estimation of the robot relative rotation among a set of images at a particular location. Therefore, we calculate the relative rotation of each image pair from the same location and compare the results against a ground truth. If the established rotation differs from the ground truth by less than 2 degrees, we consider the rotation correct. The ratio of correct estimations to a total number of comparisons is considered a success rate, which we consider a measure of the feature utility for long-term mobile robot navigation. Since the dataset contains 12 images from each location, we obtain 660 comparisons for each evaluation, which allows to establish the success rate with sufficient granularity.

(a) January, location 5 (b) Overlapped, location 5 (c) February, location 5

Figure 6.15: Imperfect alignment of the user superposed images.

The ground truth was obtained by means of a software, which allows to superimpose two images. Six persons were asked to align the images so that the same objects in both images would overlap and save the relative image coordinates. The saved values were checked for outliers and the coordinates for the individual image pairs were averaged. The robot relative heading calculated as the average of translations in image coordinates is considered a ground truth. Although the locations where the images were taken were not exactly identical and therefore it was impossible to align the images precisely (see Figure 6.15), the values given by the six users varied only by a few pixels and only two outliers were detected.

6.10 Rotation calculation

To calculate the relative rotation of the images, we have chosen two different methods. The first method closely follows a classical approach presented from

Hartley's book Multiple View Geometry in Computer Vision [168]. First, the tentative correspondences between the features from the image pair are generated. Then, a Random Sampling Consensus (RANSAC) [169] is applied to find the fundamental matrix, which defines the epipolar geometry. The fundamental matrix and camera intrinsic parameters are used to calculate the essential matrix. The essential matrix is decomposed to obtain rotation matrix and translation vector. Finally, the Euler angles are calculated from the rotation matrix.

In [49] Krajnik proposes a simplistic approach for mobile robot heading estimation. The author assume that the average distance of perceived objects is higher than the image baseline (the distance of the two places, where images were captured). Moreover, the terrain is considered locally planar, so that the robot pitch or roll is similar for both images. Under these assumptions, it is possible to calculate robot heading simply by finding a modus of horizontal displacements of the tentative correspondences. This approach can be used for landmark-based navigation as we explained in Chapter 5.

6.10.1 Feature matching

This step precedes both methods for rotation estimation. Keypoints from a pair of images corresponding from two views of the same robot position are detected by SIFT, SURF, STAR, BRISK and ORB methods. Descriptors of the vicinity of these keypoints are then calculated by SIFT, SURF, BRIEF, BRISK and ORB descriptors. Since BRIEF is only a descriptor, we combine it with the STAR and SURF detectors, as suggested by the original paper of BRIEF. However, we found BRIEF using STAR detector to be much more robust than using the SURF detector. Thus, in the rest of the Chapter we will refer to BRIEF as the BRIEF-STAR scheme.

Euclidean (in the case of SIFT and SURF floating point descriptors) or Hamming (BRIEF, ORB and BRISK binary descriptors) distances between the two sets of the descriptors are calculated. The two best matching (i.e. lowest distance between descriptors) are determined. If the distance of the closest pair is lower than 0.8 times the distance of the second closest pair, the closest image features are considered a tentative correspondence, this means that these keypoints are probably a projection of the same point in the world to the two different images.

6.10.2 Epipolar geometry based approach

This approach follows closely Hartley's book on Multiple View Geometry in Computer Vision [168]. Image keypoints are represented by homogeneous 3-vectors $\mathbf{x}$ and $\mathbf{x}'$ on the left and the right images, respectively. The corre-

spondence between them is noted $\mathbf{x}' \leftrightarrow \mathbf{x}$. This pair of keypoints is interpreted as the projections, in the two image planes, of the same point in the word. World points (keypoints in the world) are represented by homogeneous 4 vectors $\mathbf{X}$. The camera matrix P is represented by a 3×4 matrix indicating the image projection $\mathbf{x} = P\mathbf{X}$ up to a scale factor. A camera matrix with a finite projection center can be factored into $P = K[R|\mathbf{t}]$, where K is a 3×3 upper triangular calibration matrix holding the intrinsic parameters, R is a rotation matrix and t is a translation vector, which relate the camera orientation and position to a world coordinate system. They are called the external parameters of the camera.

If the origin of the world coordinate system is set on the center of one camera, then the camera matrices for the two views can be factored into $P = K[I|\vec{0}]$ and $P' = K'[R|\mathbf{t}]$, where $\vec{0}$ is the null vector and $(R, \mathbf{t})$ are the rotation matrix and translation vector between first camera center C (origin of the world coordinate system) and second camera center C'. This is called canonical form for a pair of camera matrices, describing the epipolar geometry of the Figure 6.16.

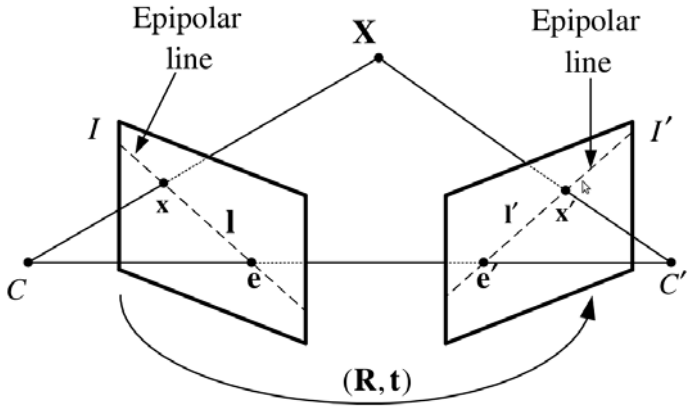


Figure 6.16: Epipolar geometry.

The fundamental matrix F can be defined as the unique 3×3 rank 2 homogeneous matrix which satisfies the epipolar constraint:

$$\mathbf{x}'^{\top} F \mathbf{x} = 0 \quad \text{for all correspondences} \quad \mathbf{x}' \leftrightarrow \mathbf{x}. \tag{6.20}$$

For any point $\mathbf{x}$ in the first image, the corresponding epipolar line is defined as $l' = F\mathbf{x}$. l' contains the epipole e', thus $e'^{\top} F\mathbf{x} = 0$ for all x (see Figure 6.16).

Equation (6.20) allows to calcutale the fundamental matrix F using the RANSAC algorithm [169]. At least 8 correspondences are required to apply RANSAC. Once F is obtained, the essential matrix can be computed using:

$$E = K'^T F K \tag{6.21}$$

where K and K' are the the intrinsic calibration matrices of the two images involved. In our case, because both images were taken with the same camera, we know that $K = K'$.

Now, rotation matrix R and translation vector $\mathbf{t}$ can be recovered from the essential matrix E on the basis of the following theorem:

Theorem 1. *Let us consider two views of the same scene taken with the same camera, let P and P' be the two associated camera matrices, then if the origin of the world coordinate system is set on the center of the first camera, i.e. $P = K[I|\vec{0}]$, and the singular value decomposition of the essential matrix is $E = UDV^\top$, there are four possible solutions for the second camera matrix P': $P'_1 = K[R_1|\mathbf{t}]$, $P'_2 = K[R_1|-\mathbf{t}]$, $P'_3 = K[R_2|\mathbf{t}]$ and $P'_4 = K[R_2|-\mathbf{t}]$ where $R_1 = UDV^\top$ and $R_2 = UD^\top V^\top$.*

A 3×3 matrix is an essential matrix E if and only if two of its singular values (from its SVD decomposition) are equal, and the third one is zero. We know that

$$E = [\mathbf{t}]_\times R = SR, \tag{6.22}$$

where $\mathbf{t} = [t_1 t_2 t_3]^\top$, R is a rotation matrix and S is a skew-symmetric matrix given by

$$S = [\mathbf{t}]_\times = \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix} \tag{6.23}$$

Let us now define the orthogonal matrix W and the skew-symmetric matrix Z as

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.24}$$

and

$$Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{6.25}$$

It can be proved that, as S is a skew-symmetric matrix, it can be decomposed as $S = kUZU^t$, where k is a real constant, and the matrix

$$u = \begin{pmatrix} u_1 & u_2 & u_3 \\ u_4 & u_5 & u_6 \\ u_7 & u_8 & u_9 \end{pmatrix} \tag{6.26}$$

is orthogonal [170]. It can be verified that $Z = diag(1, 1, 0)W$, up to sign. Then, we have that, up to a scalar, it is true that $S = Udiag(1, 1, 0)WU^\top$,

then the essential matrix can be written as:

$$E = SR = Udiag(1,1,0)\underbrace{WU^\top R}_{V^\top} \tag{6.27}$$

which is an SVD decomposition with two equal singular values and a third null value.
Now, let us suppose that the first camera is $P = [I \mid \mathbf{0}]$ and that the second camera is $P' = [R \mid \mathbf{t}]$. The SVD decomposition of the essential matrix E is given by equation (6.27), then there are two possible factorizations of E: one with $R = UWV^\top$ and the other with $R = UW^\top V^\top$. On the other hand, as $S = UZU^\top$ and the rotation matrix can be written as $R = UXV^\top$, where X is also a rotation matrix, then

$$E \ = SR = (UZU^\top)(UXV^\top) = U(ZX)V^\top. \tag{6.28}$$

Then, by 6.27 and 6.28, it can be seen that $ZX = diag(1,1,0)$, from which it can be inferred that $X = W$ or $X = W^\top$. So, the two only possible rotation matrices are $R = UWV^\top$ and $R = UW^\top V^\top$. The $\mathbf{t}$ part of the camera P' can be obtained, up to a scale factor, from $S = [\mathbf{t}]_\times$. As the Frobenius norm of $S = UZU^\top$ is $\sqrt{2}$, then if $S = [\mathbf{t}]_\times$ including scale, then $\|\mathbf{t}\| = 1$. As $S\mathbf{t} = \mathbf{0}$ then it must be that

$$\mathbf{t} = U \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \mathbf{u}_3 \tag{6.29}$$

where $\mathbf{u}_3$ is the third column of matrix U, as

$$S\mathbf{t} = UZU^tU \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = U \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \mathbf{0}. \tag{6.30}$$

But the sign of the essential matrix E cannot be determined, and so the same thing happens to the sign of $\mathbf{t}$. Then, there are four possible matrices for the P' camera:

$$P' = \left[UWV^t \mid \mathbf{u}_3\right] \text{ or } \left[UWV^t \mid -\mathbf{u}_3\right] \text{ or } \left[UW^tV^t \mid \mathbf{u}_3\right] \text{ or } \left[UW^tV^t \mid -\mathbf{u}_3\right] \quad \square$$

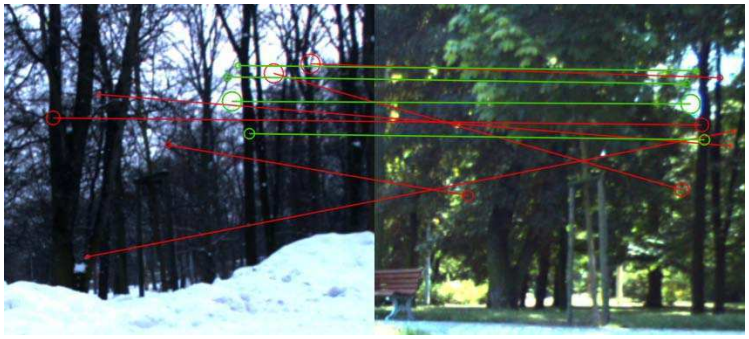
Then, there are the following four possible solutions for the second camera matrix P': $P'_1 = K[R_1|\mathbf{t}]$, $P'_2 = K[R_1|-\mathbf{t}]$, $P'_3 = K[R_2|\mathbf{t}]$ and $P'_4 = K[R_2|-\mathbf{t}]$. One of the four choices corresponds to the true configuration. Another one corresponds to the twisted pair which is obtained by rotating one of the views 180 degrees around the baseline. The remaining two correspond to reflections of the true configuration and the twisted pair.

In order to determine which choice corresponds to the true configuration, the cheirality constraint is imposed. This constraint means that the scene points should be in front of the cameras. One point is sufficient to resolve

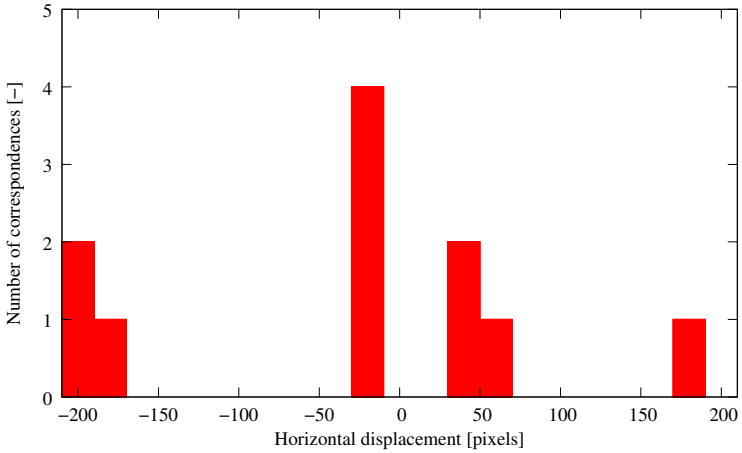
the ambiguity. The point is triangulated using the view camera pairs ($P = K[I|\vec{0}], P' = P'_i$) with $i = 1...4$ to yield the world point $\mathbf{X}$ and cheirality is tested. If $\mathbf{X}_3\mathbf{X}_4 > 0$ and $(P'_i\mathbf{X})_3\mathbf{X}_4 > 0$, then $(P = K[I|\vec{0}], P' = P'_i)$ correspond to the true configuration. Finally, the obtained rotation matrix R is decomposed into the Euler angles.

6.10.3 Histogram voting approach

This method first calculates differences in horizontal positions d_i of the tentative correspondences. After that, n histograms (denote them H_i, where $i \in \{0 \dots n-1\}$) of bin width n are calculated from values $d_i + i$. A histogram with the highest maximal bin (h_{max}) is selected and a second highest bin value (h_{sec}) of this histogram is found. The values are compared and if $h_{max} > h_{sec} + e$, the horizontal positions d_i corresponding to the highest bin are averaged. The result, which represents the displacement of the two images in pixels is then converted to robot rotation.



(a) Matched images (correct matches are green)



(b) Histogram voting

Figure 6.17: Detected correspondences and histogram voting illustration.

The method reports failure in the case of $h_{max} \leq h_{sec} + e$. Experimental results have shown, that for $e = 2$, the number of false positives is below 1%. Figure 6.17 shows the detected correspondences and the resulting histogram voting.

6.11 Results

As we have noted before, our primary measure of the feature extractor efficiency for long-term mobile robot navigation is its success rate in estimating a correct relative rotation between the dataset images. First, we want to establish a suitable number of features for correct image registration. After that, we have evaluated the success rate of the features for individual places.

We have used OpenCV (Open Source Computer Vision) [171] version 2.4.2 for programming image feature detector and keypoint descriptor functions.

6.11.1 Number of features

In the first test, we estimate the influence of the number of extracted features to the matching success rate. To extract the desired number of features, the sensitivity thresholds for the individual feature detectors has to be established.

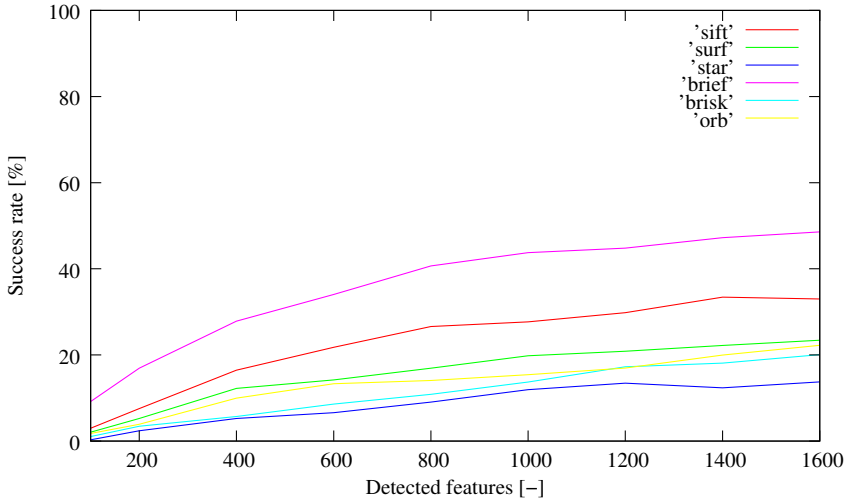
Then, the images of the dataset are processed with the particular threshold and the obtained features are used to calculate the rotations by both of the aforementioned methods. The rotations are then compared to the ground truth and the average success rate over all five locations has been calculated. The dependence of the success rate on the number of extracted features is given on Figure 6.18.

Although a higher number of extracted features means higher success rate, increasing this number beyond 1000 does not bring a substantial benefit. For this reason, in the rest of the tests we have chosen to set the detector threshold sensitivities to extract 1000 features on average.

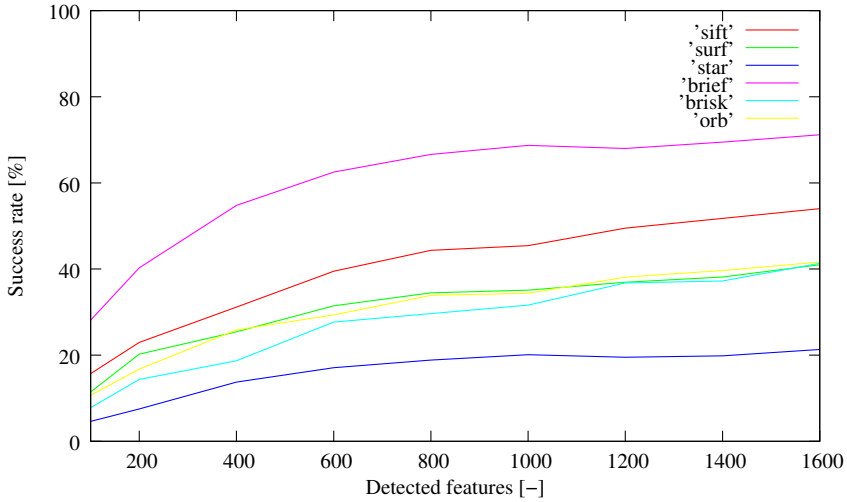
6.11.2 Success rate for individual locations

Using the aforementioned settings, we have calculated the robot relative rotations for all the 5 locations. The locations differ in the number of vegetation and buildings in the robot field of view, (see Figure 6.19).

This time, we are considering three possible outcomes of the heading estimation method. By the term “Correct”, we mean that the calculated heading



(a) Epipolar geometry based approach



(b) Histogram voting approach

Figure 6.18: Dependence of the success rate on the number of extracted features.

conforms with the ground truth. The term "Incorrect" means that the calculated heading contradicts the ground truth. The term "Failure" means that the algorithm has indicated, that the heading could not be established, e.g. due to low number of matches, or that the certainty about the heading value is low. The Tables 6.2 and 6.1 summarize the results of the epipolar geometry and the histogram voting approaches for the individual locations.

In terms of success rate, the BRIEF extractor outperforms the other algorithms. The performance of the STAR, SIFT, SURF and BRISK methods is pretty similar and ORB performance is the worst.

(a) Location I (b) Location II (c) Location III

(d) Location IV (e) Location V

Figure 6.19: View from the robot camera at different locations.

Table 6.1: Matching results for the epipolar geometry based approach.

Location	Outcome	sift	surf	star	brief	brisk	orb
I	Correct	30.3	19.6	9.8	41.6	19.6	16.6
	Incorrect	6.8	8.3	52.2	20.4	4.5	10.6
	Failure	62.8	71.9	37.8	37.8	75.7	72.7
II	Correct	13.6	11.3	5.3	29.5	9.0	8.3
	Incorrect	4.5	1.5	41.6	21.9	0.0	0.7
	Failure	81.8	87.1	53.0	48.4	90.9	90.9
III	Correct	24.2	17.4	9.8	31.8	9.8	13.6
	Incorrect	9.0	6.	21.9	22.7	3.0	4.5
	Failure	66.6	76.5	68.1	45.4	87.1	81.8
IV	Correct	31.0	22.7	13.6	51.5	13.6	18.1
	Incorrect	17.4	14.3	18.9	33.3	4.5	15.9
	Failure	51.5	62.8	67.4	15.1	81.8	65.9
V	Correct	39.3	28.0	21.2	64.3	16.6	20.4
	Incorrect	9.8	11.3	22.7	28.0	3.0	8.3
	Failure	50.7	60.6	56.0	7.5	80.3	71.2
Σ	Correct	27.6	19.8	11.9	43.7	13.7	15.4
	Incorrect	9.5	8.2	31.4	25.2	3.0	8.0
	Failure	62.6	71.7	56.4	30.8	83.1	76.5

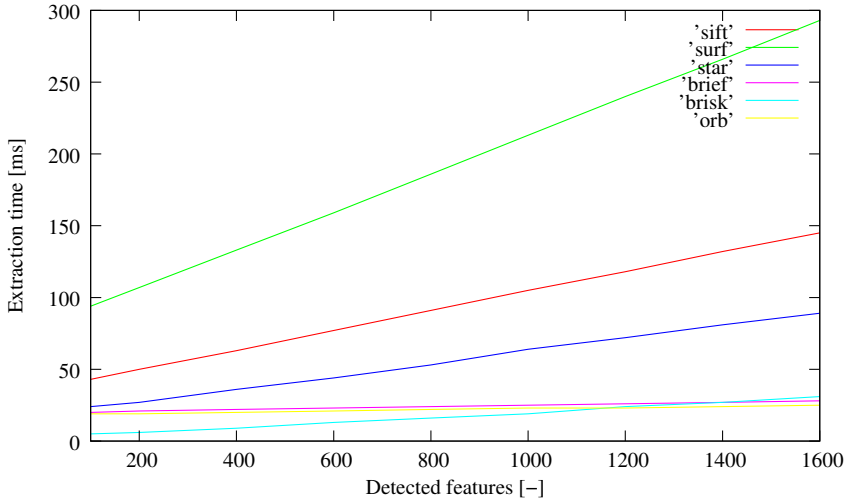
6.11.3 Feature extractor speed

A secondary measure of the feature extractor is its speed. To establish the speed, we have used the average time to extract a given number of features. Figure 6.20 shows feature extraction time depending on the number of detected features and on the detector-descriptor method used.

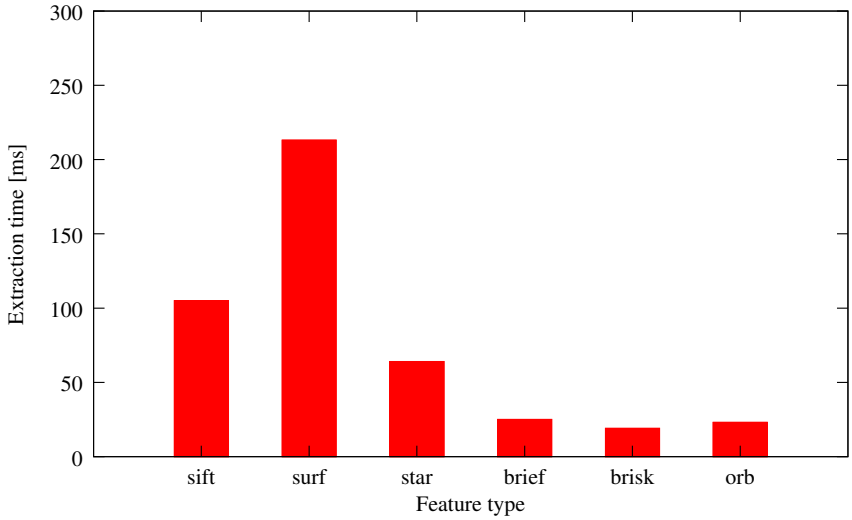
Table 6.2: Matching results for the histogram voting method.

Location	Outcome	sift	surf	star	brief	brisk	orb
I	Correct	47.7	34.8	12.8	57.5	40.9	40.9
	Incorrect	0.0	0.7	0.7	0.0	0.0	0.0
	Failure	52.2	64.3	86.3	42.4	59.0	59.0
II	Correct	24.2	16.6	10.6	46.9	16.6	16.6
	Incorrect	0.0	0.0	0.7	0.0	0.0	0.0
	Failure	75.7	83.3	88.6	53.0	83.3	83.3
III	Correct	38.6	27.2	15.9	57.5	25.7	29.5
	Incorrect	0.0	0.0	0.0	0.0	0.0	0.0
	Failure	61.3	72.7	84.0	42.4	74.2	70.4
IV	Correct	56.8	45.4	25.0	84.8	37.1	46.2
	Incorrect	0.7	0.7	1.5	5.3	0.0	2.2
	Failure	42.4	53.7	73.4	9.8	62.8	51.5
V	Correct	59.8	51.5	36.3	96.9	37.8	38.6
	Incorrect	0.0	0.0	0.0	0.0	0.0	0.0
	Failure	40.1	48.4	63.6	3.0	62.1	61.3
Σ	Correct	45.4	35.1	20.1	68.7	31.6	34.3
	Incorrect	0.1	0.2	0.5	1.0	0.0	0.4
	Failure	54.3	64.4	79.1	30.1	68.2	65.1

As it can be seen, SURF and SIFT are slower, while the other extractors exhibit a higher speed. This is not surprising since SURF and SIFT are quite outdated in comparison to nowadays feature extractor methods. A surprising fact is that the SIFT seems to be faster SURF. We assume, that it is an implementation issue of the OpenCV library version 2.4.2, which we use for the image feature extraction. It is known that in this version of the library, the SIFT implementation has been optimized from the original version, so it ends up being faster than SURF.



(a) Extraction time vs number of features



(b) Extraction of 1000 features

Figure 6.20: Feature extraction time

6.11.4 Conclusion

In this Chapter we evaluated a variety of image features detector-descriptor schemes for long-term visual navigation. We considered a scenario in which a mobile robot, running the aforementioned map-and-replay method, has to navigate a certain path over an extended (i.e. months) period of time. In the long term, the perceived scene is affected not only by lighting conditions, but also by naturally occurring seasonal variations. One of the critical questions is which kind of image preprocessing method would be most robust to changes caused by lighting and by seasonal changes. To answer this question, we use a dataset of sixty images covering an entire year of seasonal changes of a urban

park, and used this dataset to evaluate efficiency of state of the art image feature extractors. The chosen performance measure is the success rate of correct robot heading estimation. As a result of this evaluation, we conclude that a mixed approach that uses STAR based on CenSurE (Center Surround Extremas) to detect image features and BRIEF (Binary Robust Independent Elementary Features) to built the descriptor of the feature outperform the other detector/descriptor schemes. Moreover, the histogram voting method outperforms the epipolar geometry based approach in the rate of correct heading establishment and the rate of false heading.

Chapter 7

Hybrid segmentation and landmark based visual navigation

This chapter proposes a hybrid visual navigation method for mobile robots in indoor/outdoor environments. The method enables the use of both segmentation based and landmark-based navigation as elementary movement primitives to deal with different kinds of environments.

7.1 Introduction

The teach-and-replay landmark-based method presented in Chapter 5 has been successfully tested in both indoor and outdoor environments. However, this method presents some drawbacks. The robot workspace is limited only to the regions visited during the training step. The user has to guide the robot all around the entire environment before performing autonomous navigation, which may represent a very tedious process, specially in large outdoor environments. In addition, several training steps have to be performed to deal with variable environment appearance caused by varying illumination or seasonal changes. The training process requires human intervention every time any path the robot has to traverse suffers some change. Moreover, systems based on the teach-and-replay paradigm assume that the environment does not change very much between the learning and the running phase, but in the case of using visual landmarks they can vary or vanish between both phases, representing an additional problem. Finally, stability of the method presented in Chapter 5 is guaranteed for navigation trajectories that can be divided in a number straight conjoined segments. The method needs the robot to turn after certain time period in order to correct the heading. If the trajectory

includes a very long segment without turns, the accumulated odometry error can overgrow and spoil the navigation.

The motivation of the hybrid approach is to overcome the aforementioned drawbacks by merging the teach-and-replay landmark-based method with the segmentation-based approach for following paths. To do that, a topological map of the environment is defined that can be thought as a graph, where the edges are navigable path and the nodes are open areas. Using this schema, the robot can use segmentation-based navigation to traverse paths (edges of the graph), as we saw in Chapter 4, and can use landmark-based navigation to traverse open areas (nodes of the graph) where a map is needed, as described in Chapter 5.

The main advantages of this approach can be summarized as follows:

- It is not necessary to map all the environment, only the open areas where there is no naturally delimited paths. This significantly reduces the learning phase, specially in large outdoors environments.
- As the segmentation-based method is reactive and adaptive, it can deal with environmental changes (illumination, seasons, etc.). Thus, the problem of changes in the ambient is bounded to open areas (nodes of the map), where the landmarks-based method is used. By using the STAR/BRIEF schema for detecting and describing image feature in the landmarks-based method we reduce this problem, as was shown in Chapter 6.
- The convergence of the landmark-based approach needs the robot to turn periodically. If the trajectory includes a very long segment without turns, the accumulated odometry error can overgrow and spoil the navigation. However, if the robot used the segmentation-based approach to traverse this long segment using the following path method, the convergence is maintained.
- The topological map used for the hybrid approach does not need to store image features during the traversable path. As segments generally occupies most of the map, the hybrid approach needs a much more smaller map of the environment than the landmarks-based approach.

7.2 Method overview

The method begins with a mapping/learning phase where the user has to build the map by controlling the robot which will move semi-autonomously. The map of the hybrid method can be considered to be a scripted sequence of either the feature-based navigation method or the landmark-based navigation

method. To start the process, the user initiates mapping with one of the two methods, which are movement primitives for the entire system. For the case of the feature-based method, the mapping phase is completed as for the case in which this method is used standalone. Otherwise, if the user selects the landmark-based method, the robot will start to execute the path following algorithm until commanded by the user, at which point the estimated distance traveled (computed by odometry) is recorded. It is important to note that, if the length of the path is known in advance, it can be incorporated into the map and thus, there is no need to traverse this path during the learning phase. After mapping a portion of the environment, the user may select the other method for mapping until the complete map is finished.

Afterwards, for the navigation phase, the hybrid method starts executing in sequence either one of the two movement primitives available. For the case of the feature-based method, the navigation stops when the map of this subportion of the complete map is completely traversed. When executing the segmentation-based method, the robot will follow the navigable path until the estimated traveled distance is the same as the one recorded for this portion of the map.

7.3 Stability

It is possible to analyze the convergence of the hybrid navigation method as we have already done for the landmark-based navigation method. In this case, the desired path corresponds to the trained path in the case of the landmark-based method and, for the case of the segmentation-based method, to a path followed by the reactive control. Given the previous analysis of the convergence of the individual methods, it is possible to give a basic insight of the convergence of the hybrid method itself. We take the same assumptions as in Chapter 5. Now, consider that the given path is square and that the robot has to traverse it repeatedly. This square has two conjoined sides that are mapped and the other two conjoined sides that correspond to naturally delimited areas. In this case, the hybrid approach can be applied as follows.

At the beginning, the robot is placed at a random position (2D Gaussian distribution with zero mean) near the first side. The initial position uncertainty can therefore be displayed as a circle, in which the robot is found with some probability. As the first side is not naturally delimited, the landmark-based navigation method is used as the first movement primitive of the system. Because the robot senses landmarks along the segment and corrects its heading, its lateral position deviation is decreased. At the end of the segment, the uncertainty ellipse therefore becomes narrower. However, due to the odometric error, the longitudinal position error increases and therefore the ellipse is longer. The second side is traversed in the same way. When the robot

reaches the third side, the movement primitive of the system changes into the segmentation-based navigation method, since this region is naturally delimited. The path following algorithm guides the robot to the center of the path, therefore the lateral position uncertainty of the robot decreases even further. The only error in this case is the one produced by the odometry. In the last segment the robot uses the same navigation method, which therefore commands the robot to the middle of the path and the lateral position uncertainty is reduced further. Figure 7.1 shows the position uncertainty evolution in this simple symmetric case for the hybrid method.

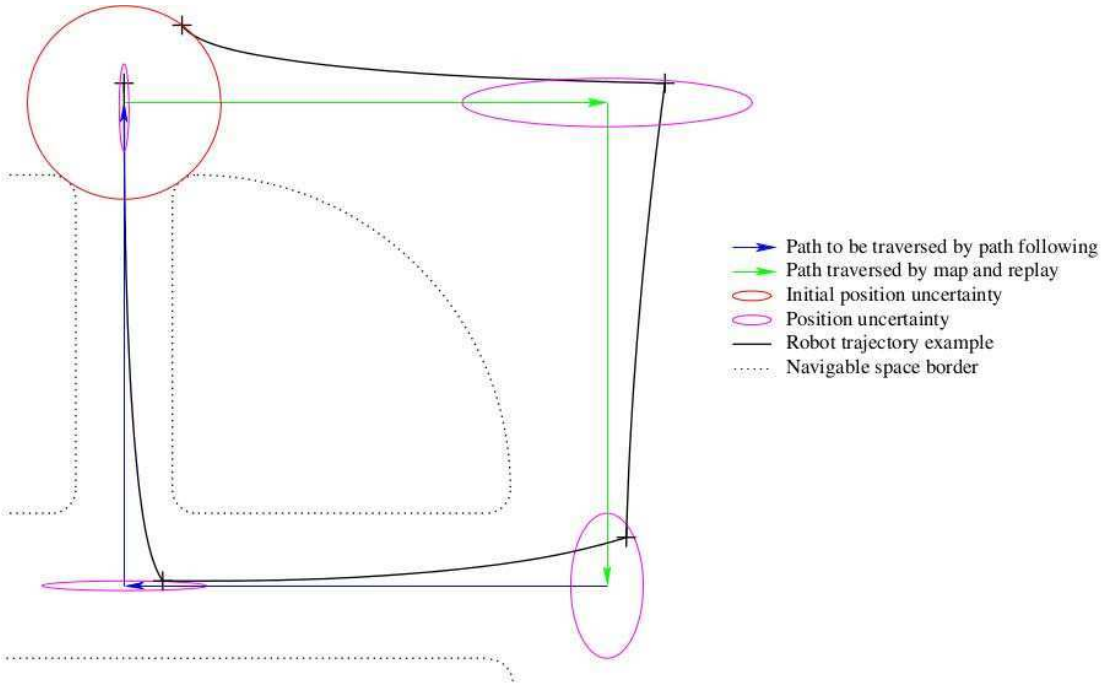


Figure 7.1: Position uncertainty evolution in a simple symmetric case.

The only difference with the stability discussed in Chapter 5 is that the path following method reduces faster the lateral error than the map-and-replay method. However, both methods have the same longitudinal error because both measure the distances by odometry. Therefore, the mathematical proof is the same as the one presented in Chapter 5.

7.4 Indoor/Outdoor experiment

The system performance for the hybrid approach has been evaluated in a indoor/outdoor environment. The experiment was performed outside and inside of the Pabellón 1 building, Ciudad Universitaria, Buenos Aires, Argentina. The robot has been taught an open area around the entrance of the building and a square path inside the hall during the training phase. In the au-

tonomous navigation phase, the change between the segmentation-based and the landmark-based method is done by odometry estimation. The approximate path is drawn in Figure 7.3 and it measures 68m. Circles indicate stating point, change point and end point respectively. The robot has been placed 0.6 m away from the starting point and requested to traverse it repeatedly 3 times. The errors for the whole path were: 12cm, 7,5cm and 3cm for each iteration, respectively. Figure 7.4 shows the results for the position errors of the experiment.

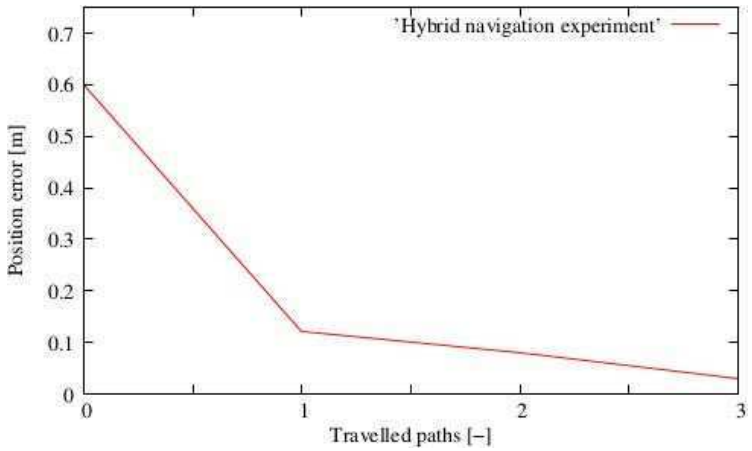
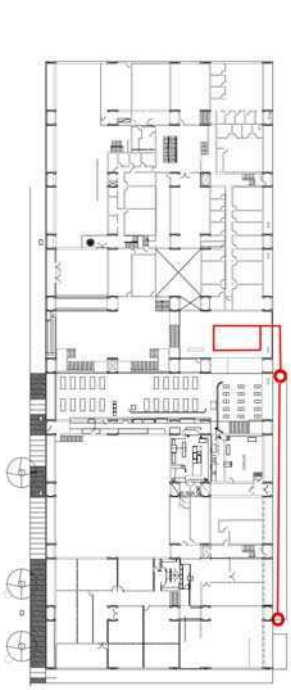
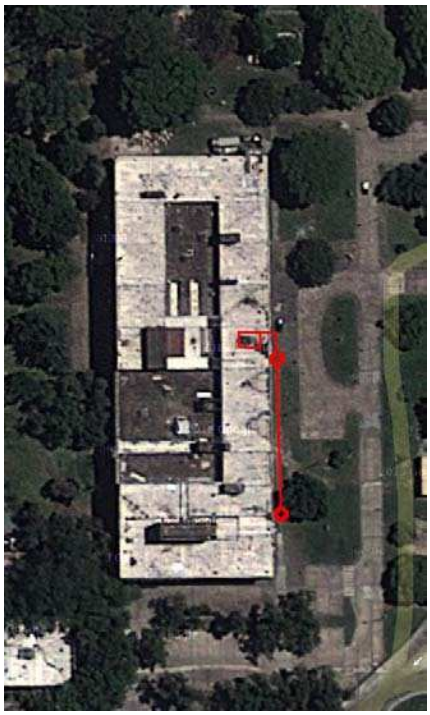


Figure 7.2: Position errors of the indoor/outdoor experiment with the hybrid method.

(a) (b) (c)



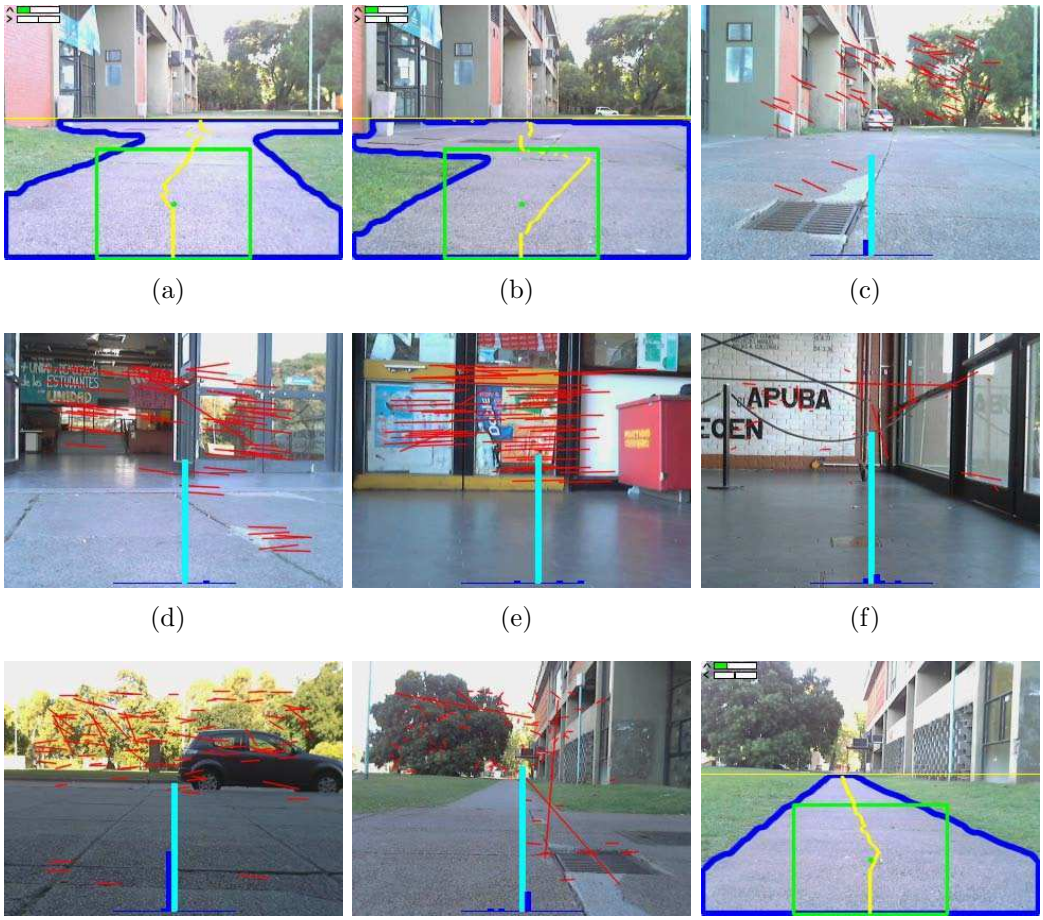
(d)



(e)

(f) (g) (h)

Figure 7.3: The indoor/outdoor experiment. The robot traversed 68m outside and inside Pabellón 1 building.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figure 7.4: Screenshots extracted from the robot: 7.4(a), 7.4(b) and 7.4(i) during the segmentation-based navigation and 7.4(c), 7.4(d), 7.4(e), 7.4(f), 7.4(g) and 7.4(h) during the landmark-based navigation.

7.5 Conclusions

In this Chapter we propose a hybrid visual navigation method for mobile robots in indoor/outdoor environments. The method enables to use both segmentation-based and landmark-based navigation as elementary movement primitives for the entire system. A topological map of the environment is defined that can be thought as a graph, where the edges are navigable path and the nodes are open areas.

As we already saw in Chapter 4 and 5, segmentation-based navigation fits very well to path following (edges) and landmark-based navigation is suitable for open areas (nodes). The presented method is robust and easy to implement and does not require sensor calibration or structured environment, and its computational complexity is independent of the environment size. The afore-

mentioned properties of the method allow even low-cost robots to effectively act in large outdoor and indoor environments. Experiments with ExaBot show these benefits in practice.

Chapter 8

Conclusions and future work

As we said in Chapter 1, the overall aim of this Thesis is to develop a new vision-based mobile robot system for monocular navigation in indoor/outdoor environments. The system is supposed to deal with real world changing conditions (illumination, seasons, etc.) and satisfy real time constraints with off the shelf sensors and computational hardware. Moreover, it should be a completely autonomous system, without dependence on external positioning or localization support or other infrastructure. To achieve the overall goal, we have proposed to achieve a set of sub-goals:

- **Review the State of the Art in Mobile Robotics**

  We have reviewed the current state of the art in Mobile Robotics. This extensive review allows us to categorize the methods associated with the main problems in Mobile Robotics. Chapter 2 addresses the Mapping, Localization and Motion Planning methods. Advantages and disadvantages of different approaches are described. This chapter also shows that although the most popular solutions for autonomous navigation use expensive sensors, mobile robot navigation is also possible with standard digital cameras.

- **Develop a new low-cost mobile robot: the ExaBot**

  In Chapter 3 we present a new low-cost mobile robot: the ExaBot. This robot was designed not for a particular task, but for a variety of research, education and popularization of science activities. To achieve this goal the robot has a novel reconfigurable hardware architecture. This architecture can be modified depending on the task at hand. It has a assorted number of sensors that include shaft encoders, telemeters, sonar, bumpers, line following and electronic compass. For motion control, a Proportional-Integral-Derivative (PID) controller was implemented. The Ziegler-Nichols method was used for tuning the PID loop and setting the proportional, integral, and derivative term of the controller. The robot's

versatility makes it an ideal platform for experimentation. The presented ExaBot was successfully as experimental platform in Chapters 4, 5 and 7 .

- **Segmentation-based navigation for outdoor environment**

  In Chapter 4 a novel autonomous vision-based navigation method for outdoor environments is presented. This method allows a mobile robot equipped with a monocular camera to travel through different naturally delimited outdoor paths. It does not require camera calibration or a priori knowledge or a map of the environment. The core of the method is based on segmenting images and classifying each segment to infer a contour of navigable space. The contour shape is used to calculate the trajectory, the linear and angular speed of the mobile robot. Regarding the segmentation step, several algorithms were evaluated on different platforms. For the case of a modern CPU, the Graph-based segmentation method proved to be the fastest. For the case of an embedded computer suited for small mobile robots, this method is not fast enough. An embedded GPU allowed the implementation of the Quick shift algorithm with execution times within required constraints and comparable to execution times of modern CPUs.

  When analyzing the path detection capability of the method, it proved to be very robust handling difficult situations associated to unstructured outdoor environments. By using an example area from which only a subset of modes with enough coverage of the Region of Interest (RoI) are used, outliers can be ignored without requiring precise placement of this region. This also allows the usage of the camera as the sole sensor, in contrast to using a laser scanner for precise path region detection. Finally, a probabilistic classification algorithm is perform to merge the segments that belong to the navigable path. For evaluation of the closed-loop performance of the algorithm, a simple control motion law was implemented which aims to maintain the robot in the middle of the detected path. By positioning the robot in different initial orientations with respect to the road, the correct behavior of the control law and the method as a whole were assessed. The simplicity of this control law, does not require the camera to be calibrated. All these features achieved allow to use this method as part of a navigation system that includes outdoor environments.

- **Landmark-based navigation for indoor/outdoor environment**

  In Chapter 5, a known landmark-based monocular navigation method is enhanced and implemented for the ExaBot. This method was proposed by Tomas Krajnik in [49]. It uses the map-and-replay technique. A topological map is built during the learning phase. This map contains visual natural landmarks information about the environment that then

is used during the autonomous navigation phase. The basic idea is to utilize image features as visual landmarks to correct the robot's heading, leaving distance measurements to the odometry. The heading correction itself can suppress the odometric error and prevent the overall position error from diverging. Experiments with ExaBot were performed to test this method in indoor and outdoor environment. The original version of the method uses a digital compass that was demonstrated to be unnecessary, since the turns of the robot can be estimated by odometry. Moreover, and more important, instead of using SURF (Speeded Up Robust Features) as visual landmarks, we propose to use START based on CenSurE (Center Surround Extremas) to detect image features and BRIEF (Binary Robust Independent Elementary Features) to describe them. In this way, a significant improvement to the known method was proposed.

- **Performace of local image features for long-term visual navigation**

  In Chapter 6 we deal with an open problem in mobile robotics which is long-term autonomy in naturally changing environments. We consider a scenario, in which a mobile robot running the aforementioned map-and-replay method has to navigate a certain path over an extended (i.e. months) period of time. In the long term, the perceived scene is not affected only by lighting conditions, but also by naturally occuring seasonal variations. One of the critical questions is which type of image preprocessing method would be most robust to changes caused by lighting and seasonal changes. To answer the question, we have used a dataset of sixty images covering an entire year of seasonal changes of a urban park and used this dataset to evaluate efficiency of state of the art image feature extractors. The chosen performance measure is a success rate of correct robot heading estimation. As a result of this evaluation, we conclude that using START based on CenSurE (Center Surround Extremas) to detect image features and BRIEF (Binary Robust Independent Elementary Features) to describe outperform the other detector/descriptor schemes.

- **Hybrid segmentation and landmark based visual navigation**

  Finally, in Chapter 7 we use the results of previous chapters to propose a novel visual navigation method for mobile robots in large indoor/outdoor environments. The method enables to use both segmentation-based and landmark-based navigation as elementary movement primitives for the entire system. A topological map of the environment is defined that can be thought as a graph, where the edges are navigable path and the nodes are open areas. As we saw in Chapter 4 and 5 , segmentation-based navigation fits very well for path following (edges) and landmark-based navigation is suitable for open areas (nodes). The presented method is

robust and easy to implement and does not require sensor calibration or structured environment, and its computational complexity is independent of the environment size. The aforementioned properties of the method allow even low-cost robots to effectively navigate in indoor/outdoor environments. Experiments with ExaBot shows these benefits into practice.

Future work

While the proposed method demonstrated to be robust and stable, there are some aspects that could be improved and further analyzed. For example, the proposed method has only been tested on terrestrial robots, but it could be extended easily to use it in other platform such as drone aerial robots. These robots usually have two built-in cameras, one facing forward and the other looking down. Thus, it would be possible to detect the path with the second camera and also detect visual landmarks with the first camera.

Other possibility to extend the results of this Thesis is to use the segmentation based navigation method to replace the human operator during the learning phase of the landmark-based navigation method. In this way, it may be achieved a method to automatically explore and map demarcated navigable areas and then use this map to replay the navigation path.

However, the most interesting issue is to extend the presented hybrid method with a position uncertainty model. The model should cover not only uncertainty of mobile robot position, but also the probability of navigation system failure depending on the environment conditions. Based on the extension of the theoretical model of the mobile robot navigation, we could advance towards real long-term mobile robot autonomy in large indoor/outdoor environments.

Bibliography

[1] J. Horáková, "Looking backward at the robot," *Research and Education in Robotics-EUROBOT 2011*, pp. 1–9, 2011.

[2] I. Asimov, "Runaround," *Astounding Science Fiction*, vol. 29, pp. 94–103, 1942.

[3] N. Nilsson, "Shakey the robot," DTIC Document, Tech. Rep., 1984.

[4] H. Moravec, *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press, 1988.

[5] S. Berman, E. Schechtman, and Y. Edan, "Evaluation of automatic guided vehicle systems," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 522–528, 2009.

[6] H. Martinez-Barbera and D. Herrero-Perez, "Development of a flexible agv for flexible manufacturing systems," *Industrial Robot: An International Journal*, vol. 37, no. 5, pp. 459–468, 2010.

[7] H. Martínez-Barberá and D. Herrero-Pérez, "Autonomous navigation of an automated guided vehicle in industrial environments," *Robotics and Computer-Integrated Manufacturing*, vol. 26, no. 4, pp. 296–311, 2010.

[8] L. Comba, P. Gay, P. Piccarolo, and D. Ricauda Aimonino, "Robotics and automation for crop management: trends and perspective," in *International Conference on Work Safety and Risk Prevention in Agro-food and Forest Systems, Ragusa, Italy*, 2010.

[9] D. Bochtis, S. Vougioukas, H. Griepentrog *et al.*, "A mission planner for an autonomous tractor," *Transactions of the ASABE*, vol. 52, no. 5, pp. 1429–1440, 2009.

[10] D. Johnson, D. Naffin, J. Puhalla, J. Sanchez, and C. Wellington, "Development and implementation of a team of robotic tractors for autonomous peat moss harvesting," *Journal of Field Robotics*, vol. 26, no. 6-7, pp. 549–571, 2009.

[11] M. Bajracharya, M. Maimone, and D. Helmick, "Autonomy for mars rovers: Past, present, and future," *Computer*, vol. 41, no. 12, pp. 44–50, 2008.

[12] R. Kerr, "Hang on! curiosity is plunging onto mars," *Science*, vol. 336, no. 6088, pp. 1498–1499, 2012.

[13] M. Chyba, "Autonomous underwater vehicles," *Ocean Engineering*, vol. 36, no. 1, pp. 1–1, 2009.

[14] H. Bendea, P. Boccardo, S. Dequal, F. Giulio Tonolo, D. Marenchino, and M. Piras, "Low cost uav for post-disaster assessment," in *Proceedings of The XXI Congress of the International Society for Photogrammetry and Remote Sensing, Beijing (China), 3-11 July 2008*, 2008.

[15] J. Han, Y. Xu, L. Di, and Y. Chen, "Low-cost multi-uav technologies for contour mapping of nuclear radiation field," *Journal of Intelligent & Robotic Systems*, pp. 1–10, 2012.

[16] S. Pedre, A. Stoliar, and P. Borensztejn, "Real time hot spot detection using fpga," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, ser. Lecture Notes in Computer Science, E. Bayro-Corrochano and J.-O. Eklundh, Eds. Springer Berlin Heidelberg, 2009, vol. 5856, pp. 595–602.

[17] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 darpa grand challenge: The great robot race*. Springer, 2007, vol. 36.

[18] S. Thrun, "Winning the darpa grand challenge," *Machine Learning: ECML 2006*, pp. 4–4, 2006.

[19] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA urban challenge: autonomous vehicles in city traffic*. Springer, 2009, vol. 56.

[20] R. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[21] iRobot Company. (2013) Roomba. [Online]. Available: http://www.irobot.com/en/us/robots/home/roomba.aspx

[22] Robomow Company. (2013) Robomow. [Online]. Available: http://www.robomow.com/

[23] Sony Company. (2013) Aibo. [Online]. Available: http://www.sony.co.uk/support/en/hub/ERS

[24] Dasarobot Company. (2013) Genibo. [Online]. Available: http://www.genibo.com/eng/index.php

[25] LEGO Company. (2013) Lego mindstorm. [Online]. Available: http://mindstorms.lego.com/en-us/Default.aspx

[26] M. Kulich, J. Chudoba, K. Kosnar, T. Krajník, J. Faigl, and L. Preucil, "SyRoTek-Distance teaching of mobile robotics," *IEEE Transaction on Education*, 2013.

[27] S. Pedre, P. De Cristóforis, J. Caccavelli, and A. Stoliar, "A mobile mini robot architecture for research, education and popularization of science," *Journal of Applied Computer Science Methods, Guest Editors: Zurada, J., Estevez*, p. 2, 2010.

[28] A. Argyros, P. Georgiadis, P. Trahanias, and D. Tsakiris, "Semi-autonomous navigation of a robotic wheelchair," *Journal of Intelligent & Robotic Systems*, vol. 34, no. 3, pp. 315–329, 2002.

[29] N. Roy, G. Baltus, D. Fox, F. Gemperle, J. Goetz, T. Hirsch, D. Margaritis, M. Montemerlo, J. Pineau, J. Schulte *et al.*, "Towards personal service robots for the elderly," in *Workshop on Interactive Robots and Entertainment (WIRE 2000)*, vol. 25, 2000, p. 184.

[30] G. Lacey and K. Dawson-Howe, "The application of robotics to a mobility aid for the elderly blind," *Robotics and Autonomous Systems*, vol. 23, no. 4, pp. 245–252, 1998.

[31] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 376–382, 1991.

[32] I. N. R. Siegwart, *Introduction to autonomous mobile robots*. The MIT Press, 2004.

[33] S. Thrun *et al.*, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, pp. 1–35, 2002.

[34] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[35] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (SLAM): Part I," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[36] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping: Part ii," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 3, pp. 108–117, 2006.

[37] J. Borenstein, H. Everett, L. Feng, and D. Wehe, "Mobile robot positioning-sensors and techniques," DTIC Document, Tech. Rep., 1997.

[38] A. Martinelli, N. Tomatis, and R. Siegwart, "Simultaneous localization and odometry self calibration for mobile robot," *Autonomous Robots*, vol. 22, no. 1, pp. 75–85, 2007.

[39] P. Lin, H. Komsuoá¸¡lu, and D. Koditschek, "Legged odometry from body pose in a hexapod robot," *Experimental Robotics IX*, pp. 439–448, 2006.

[40] J. Borenstein and L. Feng, "Measurement and correction of systematic odometry errors in mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 6, pp. 869–880, 1996.

[41] C. Tan and S. Park, "Design of accelerometer-based inertial navigation systems," *Instrumentation and Measurement, IEEE Transactions on*, vol. 54, no. 6, pp. 2520–2530, 2005.

[42] H. Chung, L. Ojeda, and J. Borenstein, "Accurate mobile robot dead-reckoning with a precision-calibrated fiber-optic gyroscope," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 1, pp. 80–84, 2001.

[43] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1. IEEE, 2004, pp. I–652.

[44] J. Civera, O. Grasa, A. Davison, and J. Montiel, "1-point ransac for extended kalman filtering: Application to real-time structure from motion and visual odometry," *Journal of Field Robotics*, vol. 27, no. 5, pp. 609–631, 2010.

[45] D. Scaramuzza, F. Fraundorfer, and R. Siegwart, "Real-time monocular visual odometry for on-road vehicles with 1-point ransac," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE, 2009, pp. 4293–4299.

[46] A. Comport, E. Malis, and P. Rives, "Accurate quadrifocal tracking for robust 3d visual odometry," in *Robotics and Automation, 2007 IEEE International Conference on.* IEEE, 2007, pp. 40–45.

[47] K. Konolige, M. Agrawal, and J. Sola, "Large-scale visual odometry for rough terrain," *Robotics Research*, pp. 201–212, 2011.

[48] G. Dudek and M. Jenkin, "Inertial sensors, gps, and odometry," 2008.

[49] T. Krajník, J. Faigl, V. Vonásek, K. Košnar, M. Kulich, and L. Přeučil, "Simple yet stable bearing-only navigation," *Journal of Field Robotics*, vol. 27, no. 5, pp. 511–533, 2010.

[50] R. Madhavan and H. Durrant-Whyte, "Natural landmark-based autonomous vehicle navigation," *Robotics and Autonomous Systems*, vol. 46, no. 2, pp. 79–95, 2004.

[51] J. Guivant, E. Nebot, and S. Baiker, "Autonomous navigation and map building using laser range sensors in outdoor applications," *Journal of robotic systems*, vol. 17, no. 10, pp. 565–583, 2000.

[52] S. Fazli and L. Kleeman, "Simultaneous landmark classification, localization and map building for an advanced sonar ring," *Robotica*, vol. 25, no. 3, pp. 283–296, 2007.

[53] T. Deissler and J. Thielecke, "Feature based indoor mapping using a bat-type uwb radar," in *Ultra-Wideband, 2009. ICUWB 2009. IEEE International Conference on.* IEEE, 2009, pp. 475–479.

[54] V. Pierlot, M. Urbin-Choffray, and M. Droogenbroeck, "A new three object triangulation algorithm based on the power center of three circles," *Research and Education in Robotics-EUROBOT 2011*, pp. 248–262, 2011.

[55] F. Thomas and L. Ros, "Revisiting trilateration for robot localization," *Robotics, IEEE Transactions on*, vol. 21, no. 1, pp. 93–101, 2005.

[56] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, "Global positioning system. theory and practice." *Global Positioning System. Theory and practice.*, vol. 1, 1993.

[57] D. Maier and A. Kleiner, "Improved gps sensor model for mobile robots in urban terrain," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* IEEE, 2010, pp. 4385–4390.

[58] S. Thrun, W. Burgard, D. Fox *et al.*, *Probabilistic robotics.* MIT press Cambridge, MA, 2005, vol. 1.

[59] P. Stepan, M. Kulich, and L. Preucil, "Robust data fusion with occupancy grid," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 35, no. 1, pp. 106–115, 2005.

[60] A. Souza and L. Goncalves, "2.5-dimensional grid mapping from stereo vision for robotic navigation," in *Robotics Symposium and Latin American Robotics Symposium (SBR-LARS), 2012 Brazilian.* IEEE, 2012, pp. 39–44.

[61] T. Marks, A. Howard, M. Bajracharya, G. Cottrell, and L. Matthies, "Gamma-slam: Using stereo vision and variance grid maps for slam in unstructured environments," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on.* IEEE, 2008, pp. 3717–3724.

[62] H. Surmann, A. Nüchter, and J. Hertzberg, "An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, no. 3, pp. 181–198, 2003.

[63] A. Kosaka and A. Kak, "Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties," *CVGIP: Image understanding*, vol. 56, no. 3, pp. 271–329, 1992.

[64] L. Latecki, R. Lakaemper, X. Sun, and D. Wolter, "Building polygonal maps from laser range data," in *International Cognitive Robotics Workshop (COGROB)*, vol. 4. Citeseer, 2004, pp. 1–7.

[65] A. Nuchter, H. Surmann, and J. Hertzberg, "Automatic model refinement for 3d reconstruction with mobile robots," in *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on.* IEEE, 2003, pp. 394–401.

[66] H. Sohn and B. Kim, "An efficient localization algorithm based on vector matching for mobile robots using laser range finders," *Journal of Intelligent & Robotic Systems*, vol. 51, no. 4, pp. 461–488, 2008.

[67] C. Belta, V. Isler, and G. Pappas, "Discrete abstractions for robot motion planning and control in polygonal environments," *Robotics, IEEE Transactions on*, vol. 21, no. 5, pp. 864–874, 2005.

[68] A. Howard and N. Roy, "The robotics data set repository (radish)," 2003.

[69] S. Se, D. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," *The international Journal of robotics Research*, vol. 21, no. 8, pp. 735–758, 2002.

[70] B. Schölkopf and H. Mallot, "View-based cognitive mapping and path planning," *Adaptive Behavior*, vol. 3, no. 3, pp. 311–348, 1995.

[71] K. Košnar, T. Krajník, and L. Přeučil, "Visual topological mapping," in *European Robotics Symposium 2008.* Springer, 2008, pp. 333–342.

[72] D. Kortenkamp and T. Weymouth, "Topological mapping for mobile robots using a combination of sonar and vision sensing," in *Proceedings of the National Conference on Artificial Intelligence*, 1995, pp. 979–979.

[73] P. Buschka and A. Saffiotti, "Some notes on the use of hybrid maps for mobile robots," in *Proc. of the 8th Int. Conf. on Intelligent Autonomous Systems*, 2004, pp. 547–556.

[74] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schmidt, *Map learning and high-speed navigation in RHINO.* Cambridge, MA, USA: MIT Press, 1998, pp. 21–52.

[75] S. Thrun, J.-S. Gutmann, D. Fox, W. Burgard, and B. J. Kuipers, "Integrating topological and metroc maps for mobile robot navigation: a statistical approach," in *Proc. of the Fifteenth National/Tenth Conf. on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, ser. AAAI '98/IAAI '98. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 989–995.

[76] M. Bosse, P. M. Newman, J. J. Leonard, and S. Teller, "SLAM in Large-scale Cyclic Environments using the Atlas Framework." *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, December 2004.

[77] G. M. Youngblood, L. B. Holder, and D. J. Cook, "A framework for autonomous mobile robot exploration and map learning through the use of place-centric occupancy grids," in *ICML Workshop on Machine Learning of Spatial Knowledge*, 2000.

[78] M. Nitsche, P. de Cristoforis, M. Kulich, and K. Kosnar, "Hybrid mapping for autonomous mobile robot exploration," in *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, vol. 1. IEEE, 2011, pp. 299–304.

[79] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT Press, 2008.

[80] J. Guivant and E. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 242–257, 2001.

[81] L. M. Paz, J. D. Tardos, and J. Neira, "Divide and Conquer: EKF SLAM in O(n)," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 1107–1120, OCT 2008.

[82] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *International Joint Conference on Artificial Intelligence*, vol. 18, 2003, pp. 1151–1156.

[83] J. Nieto, J. Guivant, and E. Nebot, "Denseslam: Simultaneous localization and dense mapping," *The International Journal of Robotics Research*, vol. 25, no. 8, pp. 711–744, 2006.

[84] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization," *Sequential Monte Carlo methods in practice*, pp. 499–516, 2001.

[85] L. Paz, P. Piniés, J. Tardós, and J. Neira, "Large-scale 6-dof slam with stereo-in-hand," *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 946–957, 2008.

[86] A. Davison, I. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 6, pp. 1052–1067, 2007.

[87] C. Estrada, J. Neira, and J. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *Robotics, IEEE Transactions on*, vol. 21, no. 4, pp. 588–596, 2005.

[88] L. Clemente, A. Davison, I. Reid, J. Neira, and J. Tardós, "Mapping large loops with a single hand-held camera," in *Robotics: Science and Systems*, 2007.

[89] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*. IEEE, 2009, pp. 83–86.

[90] S. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[91] P. Agarwal, B. Aronov, and M. Sharir, "Motion planning for a convex polygon in a polygonal environment," *Discrete & Computational Geometry*, vol. 22, no. 2, pp. 201–221, 1999.

[92] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*. IEEE, 1991, pp. 1398–1404.

[93] L. Huang, "Velocity planning for a mobile robot to track a moving target: a potential field approach," *Robotics and Autonomous Systems*, vol. 57, no. 1, pp. 55–63, 2009.

[94] S. LaValle, "Rapidly-exploring random trees a new tool for path planning," *Technical Report, Computer Science Department, Iowa State University*, 1998.

[95] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil, "Rrt-path–a guided rapidly exploring random tree," *Robot Motion and Control 2009*, pp. 307–316, 2009.

[96] T. E. Marlin and T. Marlin, *Process control: designing processes and control systems for dynamic performance*. McGraw-Hill New York, 1995.

[97] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker, "A predictive controller for autonomous vehicle path tracking," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 10, no. 1, pp. 92–102, 2009.

[98] A. Saffiotti, E. Ruspini, and K. Konolige, "Using fuzzy logic for mobile robot control," *Int. Practical Applications of Fuzzy Technologies*, pp. 185–206, 1999.

[99] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot using neural networks," *Neural Networks, IEEE Transactions on*, vol. 9, no. 4, pp. 589–600, 1998.

[100] C. Stachniss, *Robotic mapping and exploration*. Springer, 2009, vol. 55.

[101] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*. IEEE, 1997, pp. 146–151.

[102] W. Burgard, M. Moors, C. Stachniss, and F. Schneider, "Coordinated multi-robot exploration," *Robotics, IEEE Transactions on*, vol. 21, no. 3, pp. 376–386, 2005.

[103] D. Santosh, S. Achar, and C. Jawahar, "Autonomous image-based exploration for mobile robot navigation," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE, 2008, pp. 2717–2722.

[104] R. Fisher and J. Sanchiz, "A next-best-view algorithm for 3d scene recovery with 5 degrees of freedom," in *Proc. British Machine Vision Conference BMVC99*, 1999.

[105] R. Rocha, J. Dias, and A. Carvalho, "Cooperative multi-robot systems:: A study of vision-based 3-d mapping using information theory," *Robotics and Autonomous Systems*, vol. 53, no. 3, pp. 282–311, 2005.

[106] J. Borenstein and Y. Koren, "Obstacle avoidance with ultrasonic sensors," *Robotics and Automation, IEEE Journal of*, vol. 4, no. 2, pp. 213–218, 1988.

[107] H. Surmann, K. Lingemann, A. Nüchter, and J. Hertzberg, "A 3d laser range finder for autonomous mobile robots," in *Proceedings of the 32nd ISR (International Symposium on Robotics)*, vol. 19, 2001, pp. 153–158.

[108] K. Kaliyaperumal, S. Lakshmanan, and K. Kluge, "An algorithm for detecting roads and obstacles in radar images," *Vehicular Technology, IEEE Transactions on*, vol. 50, no. 1, pp. 170–182, 2001.

[109] S. May, B. Werner, H. Surmann, and K. Pervolz, "3d time-of-flight cameras for mobile robotics." in *IROS*. IEEE, 2006, pp. 790–795.

[110] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3d modeling of indoor environments," in *the 12th International Symposium on Experimental Robotics (ISER)*, vol. 20, 2010, pp. 22–25.

[111] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 4, pp. 82–90, 2006.

[112] F. Chaumette and S. Hutchinson, "Visual servo control. ii. advanced approaches [tutorial]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 109–118, 2007.

[113] F. Bonin-Font, A. Ortiz, and G. Oliver, "Visual navigation for mobile robots: a survey," *Journal of Intelligent & Robotic Systems*, vol. 53, no. 3, pp. 263–296, 2008.

[114] Z. Chen and S. Birchfield, "Qualitative vision-based path following," *Robotics, IEEE Transactions on*, vol. 25, no. 3, pp. 749–754, 2009.

[115] C. McCarthy and N. Barnes, "Performance of optical flow techniques for indoor navigation with a mobile robot," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 5093–5098.

[116] C. Chang, C. Siagian, and L. Itti, "Mobile robot monocular vision navigation based on road region and boundary estimation," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012.

[117] M. Blas, M. Agrawal, A. Sundaresan, and K. Konolige, "Fast color/texture segmentation for outdoor robots," in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 4078–4085.

[118] Khepera, Khepera II and Khepera III. (2013) K-team corporation. [Online]. Available: http://www.k-team.com

[119] Pioneer 2-DX and 3-DX. (2013) A. m. robotics. [Online]. Available: http://robots.mobilerobots.com

[120] Traxster Kit. (2013) Robotics connection. [Online]. Available: http://www.roboticsconnection.com

[121] Microcontroller PIC18F4680. (2013) Microchip. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/39625c.pdf

[122] Microcontroller PIC18F2431. (2013) Microchip. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/39616d.pdf

[123] ARM9 TS-7250. (2013) Technologic systems. [Online]. Available: http://www.embeddedarm.com/

[124] Sharp GP2D120. (2013) Sharp. [Online]. Available: http://www.technologicalarts.com/myfiles/data/gp2d120.pdf

[125] Devantech SRF05. (2013) Devantech. [Online]. Available: http://www.robot-electronics.co.uk/htm/srf05tech.htm

[126] Allegro ACS712. (2013) Allegro. [Online]. Available: http://www.allegromicro.com/Products/Current-Sensor-ICs/Zero-To-Fifty-Amp-Integrated-Conductor-Sensor-ICs/ACS712.aspx

[127] STMicroelectronics L298. (2013) Stmicroelectronics. [Online]. Available: http://pdf1.alldatasheet.es/datasheet-pdf/view/22437/STMICROELECTRONICS/L298.html

[128] T. Pire, P. De Cristóforis, M. Nitsche, and J. Jacobo Berlles, "Stereo vision obstacle avoidance using depth and elevation maps," in *IEEE VI RAS Summer School on "Robot Vision and Applications"*, Santiago, Chile, Dec. 2012.

[129] Vision en Robótica. (2013) Departamento de Computación, FCEN-UBA. [Online]. Available: http://www-2.dc.uba.ar/materias/visrob/

[130] P. De Cristóforis, S. Pedre, M. Nitsche, T. Fischer, F. Pessacg, and C. Di Pietro, "A behavior-based approach for educational robotics activities," *IEEE Transaction on Education*, vol. 56, pp. 61–66, 2013.

[131] W. Yanqing, C. Deyun, S. Chaoxia, and W. Peidong, "Vision-based road detection by monte carlo method," *Information Technology Journal*, vol. 9, no. 3, pp. 481–487, 2010.

[132] C. Rasmussen, Y. Lu, and M. Kocamaz, "Appearance contrast for fast, robust trail-following," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on.* IEEE, 2009, pp. 3505–3512.

[133] H. Kong, J. Audibert, and J. Ponce, "General road detection from a single image," *Image Processing, IEEE Transactions on*, vol. 19, no. 8, pp. 2211–2220, 2010.

[134] P. Moghadam, J. Starzyk, and W. Wijesoma, "Fast vanishing-point detection in unstructured environments," *Image Processing, IEEE Transactions on*, vol. 21, no. 1, pp. 425–430, 2012.

[135] M. Nieto and L. Salgado, "Real-time vanishing point estimation in road sequences using adaptive steerable filter banks," in *Advanced Concepts for Intelligent Vision Systems*. Springer, 2007, pp. 840–848.

[136] T. Kuhnl, F. Kummert, and J. Fritsch, "Monocular road segmentation using slow feature analysis," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*. IEEE, 2011, pp. 800–806.

[137] I. Ulrich and I. Nourbakhsh, "Appearance-based obstacle detection with monocular color vision," in *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2000, pp. 866–871.

[138] P. Felzenszwalb and D. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[139] Y. Wang, S. Fang, Y. Cao, and H. Sun, "Image-based exploration obstacle avoidance for mobile robot," in *Control and Decision Conference, 2009. CCDC'09. Chinese*. IEEE, 2009, pp. 3019–3023.

[140] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, "Self-supervised monocular road detection in desert terrain," in *Proc. of Robotics: Science and Systems (RSS)*, 2006.

[141] S. Ettinger, M. Nechyba, P. Ifju, and M. Waszak, "Vision-guided flight stability and control for micro air vehicles," *Advanced Robotics*, vol. 17, no. 7, pp. 617–640, 2003.

[142] A. Neto, A. Victorino, I. Fantoni, and D. Zampieri, "Robust horizon finding algorithm for real-time autonomous navigation based on monocular vision," in *Intelligent Transportation Systems (ITSC), 2011 14th International IEEE Conference on*. IEEE, 2011, pp. 532–537.

[143] M. Sezgin *et al.*, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic imaging*, vol. 13, no. 1, pp. 146–168, 2004.

[144] A. Vedaldi and S. Soatto, "Quick shift and kernel methods for mode seeking," in *Computer Vision - ECCV 2008*, ser. Lecture Notes in Computer Science, D. Forsyth, P. Torr, and A. Zisserman, Eds. Springer Berlin / Heidelberg, 2008, vol. 5305, pp. 705–718.

[145] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[146] Y. Sheikh, E. Khan, and T. Kanade, "Mode-seeking by medoidshifts," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on.* IEEE, 2007, pp. 1–8.

[147] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 5, pp. 603 –619, may 2002.

[148] B. Fulkerson and S. Soatto, "Really quick shift: Image segmentation on a gpu," in *ECCV 2010 Workshop on Computer Vision on GPUs (CVGPU2010)*, 2010.

[149] B. Fulkerson and A. Vedaldi, "Really quick shift: Image segmentation on a gpu," 2010, version 0.2. [Online]. Available: http://vision.ucla.edu/~brian/gpuquickshift.html

[150] K. Kosnar, T. Krajnik, and L. Preucil, "Visual topological mapping," in *European Robotics Symposium 2008*, ser. Springer Tracts in Advanced Robotics, H. Bruyninckx, L. Preucil, and M. Kulich, Eds. Springer Berlin / Heidelberg, 2008, vol. 44, pp. 333–342.

[151] T. Krajník, "Large-scale mobile robot navigation and map building," *PhD Thesis*, 2011.

[152] E. Royer, M. Lhuillier, M. Dhome, and J. Lavest, "Monocular vision for mobile robot localization and autonomous navigation," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 237–260, 2007.

[153] Z. Chen and S. Birchfield, "Qualitative vision-based mobile robot navigation," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE, 2006, pp. 2686–2692.

[154] D. G. Lowe, "Object recognition from local scale-invariant features," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 2. Ieee, 1999, pp. 1150–1157.

[155] T. Lindeberg, "Scale-space theory: A basic tool for analyzing structures at different scales," *Journal of applied statistics*, vol. 21, no. 1-2, pp. 225–270, 1994.

[156] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[157] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *Computer Vision–ECCV 2006*, pp. 404–417, 2006.

[158] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[159] M. Agrawal, K. Konolige, and M. Blas, "Censure: Center surround extremas for realtime feature detection and matching," *Computer Vision–ECCV 2008*, pp. 102–115, 2008.

[160] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," *Computer Vision–ECCV 2010*, pp. 778–792, 2010.

[161] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2548–2555.

[162] E. Mair, G. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," *Computer Vision–ECCV 2010*, pp. 183–196, 2010.

[163] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," *Computer Vision–ECCV 2006*, pp. 430–443, 2006.

[164] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: an efficient alternative to sift or surf," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2564–2571.

[165] C. Harris and M. Stephens, "A combined corner and edge detector," in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50.

[166] P. L. Rosin, "Measuring corner properties," *Computer Vision and Image Understanding*, vol. 73, no. 2, pp. 291–307, 1999.

[167] J.-Y. Bouguet. (2004) Camera calibration toolbox for matlab. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/

[168] R. Hartley, *Multiple view geometry in computer vision*. Cambridge University Press, 2008.

[169] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[170] G. H. Golub and C. F. Van Loan, *Matrix computations*. Johns Hopkins University Press, 1996, vol. 3.

[171] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.