Nomadic Surveillance Camera

Nuno Ribeiro, Ricardo Ferreira, José Gaspar
Instituto Superior Técnico / UTL, Lisbon, Portugal
nuno.ribeiro@ist.utl.pt, jag@isr.ist.utl.pt, ricardo@isr.ist.utl.pt

ABSTRACT

Personal, inexpensive, easy-to-use, teleoperated, autonomous, mobile robots are starting to be a reality and are expected to be widespread within some few years. We present a proper methodology to control the robot over a network and make it return to its initial position. In this thesis we detail the base components forming the robot in terms of hardware and software, show navigation experiments based in Monocular SLAM, and propose a methodology to regulate the accumulated odometry error associated with map-less Monocular SLAM. We also present strategies for homing a robot to the point from where it started being teleoperated.

Keywords: Teleoperation, MonoSLAM, Homing, Wireless IP Camera, Arduino, Car-like Robot.

I. INTRODUCTION

Current personal robots are starting to have affordable prices. Those robots can be vacuum cleaners such as the iRobot Roomba, can be telepresence robots, as the Anybots' QA, or can be simply Mobile Webcams, such as the WowWee's Rovio. Most of these robots have in common the combination of mobile robotics, video cameras and wireless communications. In this work we propose using the wireless network of surveillance cameras as a basis to build networked mobile robots. We explore the possibility of having a teleoperated robot working indoors, allowing the user to check up its home even if he is far away.

A. Related Work

We calibrated our camera using Bouguet's toolbox [1], in order to extract its intrinsic parameters, which are essential to determine 3D points projection into the 2D image plane. To understand the kinematics involving our robot we resorted "Car Automatic Parking" [4], which presents the Kinematic model for a car-like robot. From "Calibrating a Network of Cameras based on Visual Odometry" [7] we extracted information about decomposing a non-singular matrix using Gram-Schmidt, since we required a strategy to decompose a projection matrix in order to determine the camera localization. "Control an RC car from a computer over a wireless network" [3] presents software and documentation for teleoperation, and suggests homing as an important topic for future work, without developing it on their project. We took advantage of some of their software, more precisely, the communications software used to require a video feed from the camera and to send instructions that can be interpreted inside the camera. The article "1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry" [2] shows, conceptually, that is possible to have monocular vision, in our robot, in order to do visual odometry. The proposed algorithm also allows self-localization and robust mapping.

B. Problem Formulation

In this work we propose assembling a car-like robot using a IP wireless camera and onboard processing in order to have a setup, capable of being controlled over a network which can localize itself without any previous knowledge of the environment. This robot must be able to return to its initial position autonomously. The objectives of this work are threefold: (i) assemble one wireless camera and one microcontroller on a car; (ii) develop the required communication software between the different processing modules (PC, IP wireless

camera, onboard microcontroller); (iii) design a homing strategy which enables the robot to return autonomously to its initial position, using a Monocular SLAM solution.

C. Thesis Structure

Section 1 introduces the objectives of this work, in particular, presents a short discussion on the state of the art and our expectations with the project at hand. In Section 2, we present our setup for experiments and describe the Hardware and Software components involved. Section 3 presents the theory behind our camera and the Kinematic model of the vehicle. Section 4 describes the possible strategies for homing and self-localization localization. Experimental results are shown and explained in Section 5. Section 6 summarizes the work done and highlights the main achievements and what we learned with this work. Moreover, this section proposes further work to extend and improve the strategies and experiments described in this document.

II. Hardware and Software Setup

The first step of our work consists of creating a reliable setup. The conceptual design of our system is shown in fig.1.

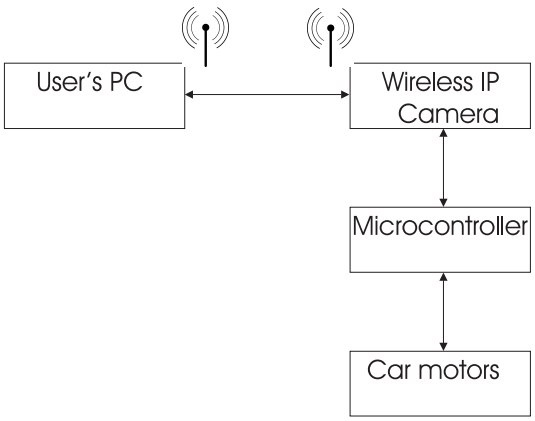


Fig. 1. Conceptual design of our system.

A. Hardware components

To build the proposed setup, hardware was selected as much as possible of-the-shelf. All of these components have strict specifications which are described in the design subsection. Every element is important for the right behavior of the complete system. The body and DC motors of the car are from a commercial R/C car. The wireless IP camera used is an Axis 207w, which is a camera built for the surveillance market. The microcontroller is an Arduino Uno. However, the Arduino can not output sufficient current to power up the motors. One has to use the Arduino Motor Shield mounted in the Arduino Uno. The used hardware components are shown in fig.2.

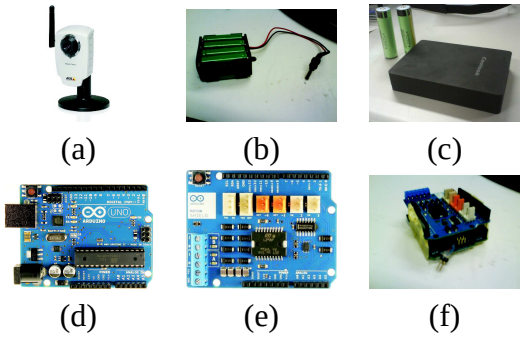


Fig. 2. (a) Axis 207w camera. (b) 8 $\times$ 1.2V AA battety pack. (c) Lithium batteries and USB box. (d) Arduino microcontroller. (e) Arduino Motor Shield. (f) Arduino and Motor Shield assembled

B. Design

One operational step consists in establishing the connection between the camera mounted in the RC car and the user's computer and creating a way to send commands from the client's computer to the car. We need a way to send commands from the camera to the microcontroller and a way to send commands from the microcontroller to the car motors. No less important, is getting a real time video feed from the camera to the user's computer (to allow real time driving) and creating a friendly graphical user interface for teleoperation. A more specific design of our project at this stage is shown in fig.3.

1) Hardware: The hardware functionality is a key aspect of this project. The design of it is divided by the following elements.

a) Wireless IP camera: The camera used in this project is an Axis 207w. The processor within the camera allows executing a Linux Operating System. The I/O terminal connector composes four pins. Two pins are for power sourcing and the others two are a digital input and a digital output. The digital output has a minimum output interval of 10 ms. The camera has two main functions. The first

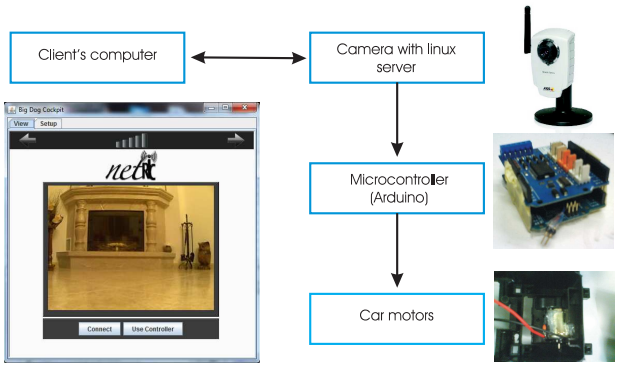


Fig. 3. Design of project at teleoperation stage.

is making available a video feed that can be sent to a user computer via wireless. Then, it is possible to open and process each frame in the user's computer. The second main function is to run a TCP/IP server, so a user can get connected and send commands to control the car. This is possible, since the camera runs a Linux OS, which allows us running a TCP server and activating/deactivating the digital output of the camera and that is the door from where we can send signals to the microcontroler, for speed and steering controls. The camera requires a constant voltage within the range of 4.9V to 5.1V.

b) Arduino Microcontroller: In order to control the motors, we need a programmable microcontroler, which controls the speed and direction of rotation of each motor. We can control the speed of rotation with a Pulse Width Modulated (PWM) signal, supplied by the microcontroller. Besides, the camera has a minimum output period of 10 ms, which is not adequate for supplying a PWM signal. Instead, the Arduino receives a message from the IP Camera and translates it into the appropriate signals for the steering and speed control. The frequency of the used PWM signal is 32KHz. The PWM configuration also allows changing the signal duty cycle, which translates in middle voltage reference values for the motors. This way, one can speed up or turn the robot to levels between its maximum and minimum.

c) Arduino Motor Shield: Motor Shield lets us drive two DC motors with the Arduino board, controlling the speed and direction of each one independently. This shield has two separate channels and each one uses 4 of the Arduino pins to drive or sense the motor. Each channel supplies a maximum current of 2A. In this way, we can control the direction of each motor by setting HIGH or LOW the direction pins. We can control the speed by varying the PWM duty cycle values.

d) DC motors: The DC motors are controlled by the Arduino (microcontroller) plus Motor Shield. The current sent to the motors is a high frequency PWM signal. In that way, we avoid forcing the motors to their maximum, which is extremely relevant when talking about the front (directional) motor, because this one should not saturate in the maximum turn angle. If that happens, i.e. the motor constantly forcing the directional end of course, the car will not turn more and the motor can be damaged.

e) Battery: The car can work, for some time, with a pack of 8 $\times$ 1.2V (9.6V) AA batteries at 1000mAh. With this power source, the camera is powered up by the Arduino, which means that all power required to feed the robot passes through Arduino. This leads to substantial heating of some sections of the Arduino board.

To improve reliability and time of operation of our robot, we decided to use another power source. Using a box with 2 USB output ports, with four Panasonic NCR18650B lithium-ion batteries inside (each one deploys a current of 2.5 A and has a capacity of 3400 mAh), it was possible to split the Arduino, Motor Shield and motors power source from the camera power source. This way, the camera is no longer powered by the electric system in Arduino, meaning the noise introduced by the Arduino Motor Shield and motors no longer reaches the camera. With this strategy, we can use our robot for a number of hours without having to recharge. This strategy is shown in fig.4. A picture of our robot is in fig.5.

2) Comunications and Software: The hardware set is the foundation of our work. Software is required to make more elaborated integration tasks. These involve data processing and communications protocols, which will be explained in the next sections. The architecture is centered on a server on the camera which receives digital commands from the user's PC program and translate them into a signal, which is interpreted by the Arduino program. The user also requests a video feed for image processing,

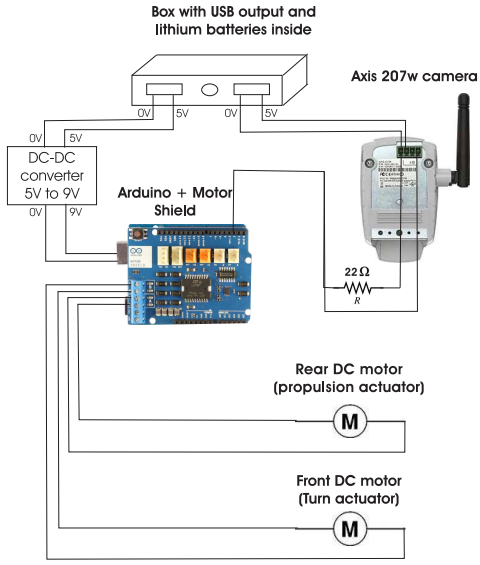


Fig. 4. Electric scheme of robot using lithium batteries.

Fig. 5. Robot setup - final version after assembling all hardware components.

which is sent by the camera.

a) User PC: To fulfill the need for a user program that allows us to communicate to the server and get the video feed from the camera, we decided to use a Java program made by [3], adapted to our hardware and protocols. Once started, this program opens a GUI window, where we can see the camera image feed, retrieved with HTTP requests, and control the car.

To adapt the program to our needs, we made some improvements and created new methods. In order to prevent the case where the user is controlling the car and decides to close the application without stopping the car, we make him stop when the red cross button is pressed, by sending the correct commands to the robot, before the application gets closed. We created a method to allow a request of the current frame from an outside program (which is useful from the point we start to use MATLAB for real-time experiments). Also, a method to be called when we want to send commands to the robot from MATLAB, instead from keyboard and a method to save the video feed into the hard disk, in order to create datasets which can be processed in MATLAB, without the real-time constraints. For efficiency reasons, this last method is never executed in real-time experiments.

b) Connection between user's PC and camera (video feed): If the request for a MJPEG video within the Java program was successful, the server in the camera returns a continuous flow of JPEG files. Note that one can see the video feed of the camera at any time by introducing the ip adress associated with the camera in an internet browser.

c) Connection between user's PC and camera (commands): To send commands to the camera, in order to control the car, we applied a Client/Server architecture using a connection between TCP/IP sockets. In the Java program running in the user's PC, when the connect button is pressed, a request to start a TCP connection is made to the server running in the camera. After the camera accepts this connection, the user is able to start sending commands. A key listener interface is listening the keyboard. Once a key is pressed, an event is triggered and an integer, corresponding to the command required is sent to the camera server by TCP/IP connection. Basically, the server program functionality boils down to open a socket, listening to it, accepting a client who is trying to establish connection and interpret the received commands.

d) Connection between camera and microcontroller: When a connection with a user is accepted, the server in the camera receives message commands, from which it produces signals to send to the microcontroller, corresponding to the right instruction accordingly to the Arduino protocol implemented by us. The signal comes out of the digital output in the camera and is received in the Arduino by one of the digital I/O ports. Arduino detects the right control command to apply to the motors by counting the number of transitions from LOW to HIGH sent by the camera output. After an end of

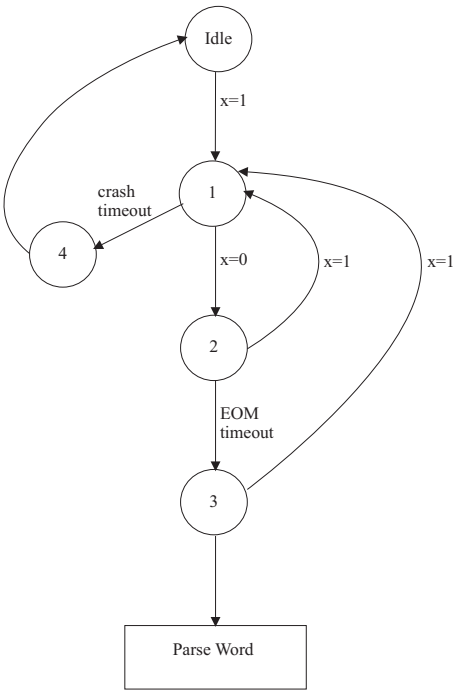


Fig. 6. Arduino program to receive one command, where x represents the digital input value received from the camera. State 1 corresponds to wait-for-transition from HIGH to LOW. State 2 corresponds to wait-for-transition from LOW to HIGH. State 3 occurs when a completed command was received. State 4 is reached when a crash was detected due to non-detected transition from HIGH to LOW, within a predefined period of time.

message timeout is achieved, the program prepares itself to receive a new instruction. If the camera crashes and the digital output is HIGH, the Arduino program will recognize the crash and will return to the initial state. After Arduino receives a complete instruction, it will update the signals which are constantly controlling the motors. The Arduino protocol established in its program is shown in fig.6.

III. Camera and Car Modeling

In this section we detail how 3D points in a generic reference frame translate into 2D points in an image plane and how the camera parameters can be estimated. We also describe the continuous kinematic model of the car and its discrete version, necessary to be processed in a computer.

A. Camera Calibration

In order to determine the pixel corresponding to a 3D point, one needs to determine the extrinsic and intrinsic parameters of the camera. The extrinsic parameters are composed by a rotation matrix and a translation vector, which allow changing from the object reference frame to the camera reference frame. In the case of non-varying optics, the intrinsic parameters are fixed, and to determine them one can use Bouguet's calibration toolbox [1].

In our case we want to determine both the intrinsic and extrinsic parameters. In other words we want to estimate the camera projection matrix P. In order to do that, we need to know the 2D coordinates (m) of feature points and their corresponding 3D coordinates (M). From [6], we can write

$$\lambda m = PM \tag{1}$$

where

$$M = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

and λ is a scale factor that equals depth in 3D ($\lambda = Z$). So, from eq.1 we have

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} P_{1:}M \\ P_{2:}M \\ P_{3:}M \end{bmatrix} \quad . \tag{3}$$

From here we can observe $\lambda = P_{3:}M$. Knowing this, we can easily write

$$\begin{cases} P_{1:}M - uP_{3:}M = 0 \\ P_{2:}M - vP_{3:}M = 0 \end{cases} \tag{4}$$

Putting eq.4 in matrix form, gets

$$\begin{bmatrix} M_1^T & \vec{0} & -u_1 M_1^T \\ \vec{0} & M_1^T & -v_1 M_1^T \end{bmatrix} \begin{bmatrix} P_{1:}^T \\ P_{2:}^T \\ P_{3:}^T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5}$$

and for N points:

$$\begin{bmatrix} M_1^T & \vec{0} & -u_1 M_1^T \\ \vec{0} & M_1^T & -v_1 M_1^T \\ M_2^T & \vec{0} & -u_2 M_2^T \\ \vec{0} & M_2^T & -v_2 M_2^T \\ \dots & \dots & \dots \\ M_n^T & \vec{0} & -u_n M_n^T \\ \vec{0} & M_n^T & -v_n M_n^T \end{bmatrix} \begin{bmatrix} P_{1:}^T \\ P_{2:}^T \\ P_{3:}^T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} \quad . \tag{6}$$

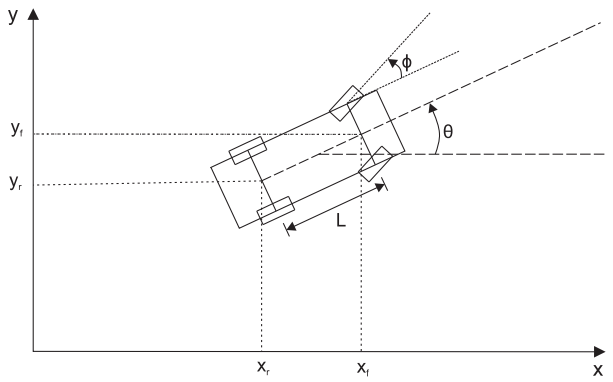


Fig. 7. Parameters to describe the car's model and motion.

We call the first matrix A and the second one π. We want minimize the norm of the residual $r = M\pi$. To solve this, π must be the normalized eigenvector associated to the smallest eigenvalue of $A^T A$. To compute the required eignvalue, one can use SVD decomposition $[U, S, V] = svd(A)$ where S is a diagonal matrix of the same dimension as A, with non-negative diagonal elements, U and V are unitary matrices, so that $A = U * S * V^T$. In fact, V contains the ordered eigenvectors corresponding to the ordered eigenvalues in S. The projection matrix elements we are looking for are in the eigenvector corresponding to the lowest eigenvalue. Then we must order them as a matrix 3×4.

B. Kinematic model

The vehicle we use in this project has rear wheel drive and is able to change direction by turning its front wheels. The points that can be reached by the vehicle are path dependent. One needs for a nonholonomic kinematic model to describe the system and predict motion.

C. Continuous model

Using the kinematic model presented at [4], we can write the kinematic equations related to the central point of the front axis as:

$$\begin{cases} \dot{x}_f = v\cos(\theta + \phi) \\ \dot{y}_f = v\sin(\theta + \phi) \\ \dot{\theta} = v\frac{\sin(\phi)}{L} \end{cases} . \tag{7}$$

where v is the linear velocity inputted to the car.

D. Discrete model

In order to simulate the kinematic model on a computer, we have to translate its equations to discrete time. To achieve that, we make first order derivative using Leapfrog integration [10], which gives us a good approximation.
Considering N to be the timestep and n to be the actual discrete time instant, for the velocities, we get:

$$\begin{cases} \dot{\theta}\,[n] = \frac{v[n]\sin(\phi[n])}{L} \\ \dot{x}_f\,[n] = v\,[n]\cos\left(\theta\,[n] + \phi\,[n] + \frac{N}{2}\dot{\theta}\,[n]\right) \\ \dot{y}_f\,[n] = v\,[n]\sin\left(\theta\,[n] + \phi\,[n] + \frac{N}{2}\dot{\theta}\,[n]\right) \end{cases} \tag{8}$$

for the positions, we get

$$\begin{cases} \theta[n] = \theta[n-1] + N\dot{\theta}[n] \\ x_f[n] = x_f[n-1] + N\dot{x}_f[n] \\ y_f[n] = y_f[n-1] + N\dot{y}_f[n] \end{cases} \tag{9}$$

IV. Navigation and Control

We present in this section some strategies in order to execute the homing procedure.

A. MonoSLAM

In order to use the camera as a sensor from where we can extrapolate data to compute the pose of the vehicle, we decided to use the MonoSLAM [2]. By using MonoSLAM we obtain the vehicle position and orientation in a 3D general frame and the location of important features from where the robot knows where it is located. The MonoSLAM explained in [2] is an algorithm, which consists in an Extended Kalman Filter (EKF) with a special Random Sample Consensus (RANSAC) embedded. Initially, the algorithm assumes that an a priori probability distribution over the model parameters is known which allows reducing the computational cost in calculating the model. The EKF makes a pose prediction of the vehicle. The update step is made by using the most voted hypothesis in RANSAC which generates hypothesis from candidate features matches. To detect local features in the images a corner extraction procedure is used.

The state vector ${}^W\hat{x}_k$ is composed by a set of camera parameters ${}^W\hat{x}_{C_k}$ and map parameters ${}^W\hat{y}_{C_k}$. All these parameters are referred to a static frame

$${}^W\hat{x}_k = \begin{pmatrix} {}^W\hat{x}_{C_k} \\ {}^W\hat{y}_{C_k} \end{pmatrix} . \tag{10}$$

The estimated map ${}^{W}y$ is composed of n point features ${}^{W}y_i$ and ${}^{W}y = \begin{pmatrix} y_1^{WT} & \dots & y_n^{WT} \end{pmatrix}^{\mathrm{T}}$. In our case, we use MonoSLAM as a visual odometer and we are interested in getting the localization of the camera and the 3D features information. We get this by accessing the Extended Kalman Filter state vector ${}^{W}\hat{x}_k$.

B. Autonomous Homing

In this section, we describe the homing strategies and discuss the advantages and drawbacks of each one. The complete project design is shown in fig.8.

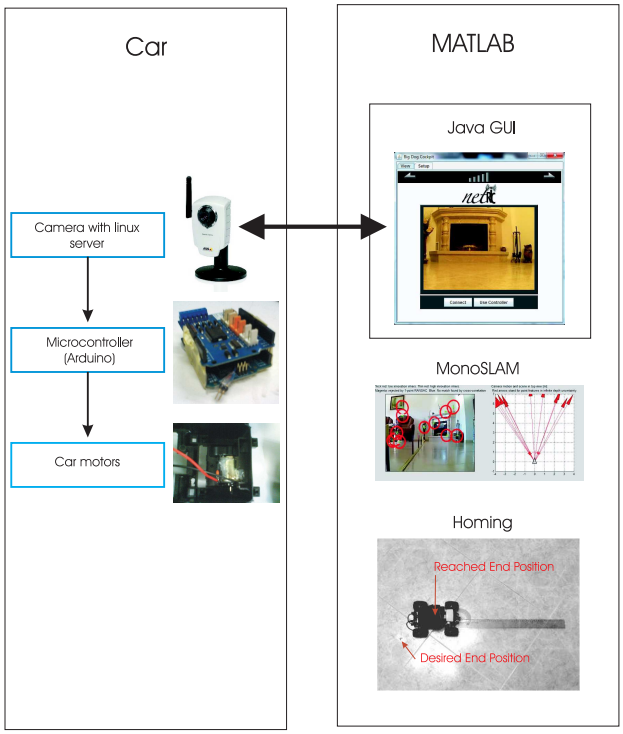


Fig. 8. Design of project at Homing stage.

1) Open Loop Homing: Following the objectives for this master's thesis, we want to teleoperate the vehicle for some time, then give the order to initiate the homing routine, which leads the vehicle to reverse and come back by a similar way it came in. The path is stored by stacking the robot position in every iteration. This is possible, since we can access the state vector from the EKF embedded in MonoSLAM, from which we determine the translation and rotation from camera to world reference frame.

One extracts from the vector *state* the position of the car

$$ {}^{W}t_C = state(1:3) = [x\ y\ z]^T \tag{11}$$

and compute its rotation matrix in the world frame

$$ {}^{\mathrm{W}}\mathrm{R_C} = \mathrm{q2r}(\mathrm{state}(4:7)) \tag{12}$$

where $q2r$ denotes the conversion for a quaternion to a rotation matrix. As described in [9], one can compute c_2 by:
$c_2 = \sqrt{{}^{\mathrm{W}}\mathrm{R_C}(1,1)^2 + {}^{\mathrm{W}}\mathrm{R_C}(1,2)^2}$, and determine θ_2, the rotation angle about the z-axis, with:

$$ \theta_2 = \mathrm{atan2}(-{}^{\mathrm{W}}\mathrm{R_C}(1,3), \mathrm{c}_2). \tag{13}$$

Notice that we are using 3D rotations when we have a 2D problem. The reason behind this, is that the used MonoSLAM program was made for 3D context and we tried to use as much as possible what was already developed. After the user activates the homing procedure, the car reverses and the controller actions take place. Its reference orientation is calculated as

$$ \theta_{ref} = a\tan 2\left(\frac{y_{ref} - y_{actual}}{x_{ref} - x_{actual}}\right). \tag{14}$$

To calculate the vehicle's pose in each iteration of homing in order to choose the right command to send to the car, a routine using the kinematic of the vehicle presented in subsection III-B is used.

This strategy is used in real-time experiments, since the features matching in MonoSLAM running in MATLAB fail after abrupt movements.

2) Visual Odometry Based Homing: Like before, in a first stage, we have MonoSLAM running, stacking the camera's localization at every iteration. When homing is activated, the vehicle reverses and, using the controller, it follows the stored path. Its pose is retrieved from MonoSLAM at every iteration. This strategy is theoretically better than using the first one, since it uses a sensor at every stage of homing, despite the consecutive error accumulation caused by visual odometry.

3) Map Based Homing: With this strategy, we also store the 3D features coordinates when the vehicle is being teleoperated, building that way a map. In the homing procedure, we compute the actual vehicle's pose, by finding the camera matrix P and decompose it to the form $K[{}^{C}R_{W}|{}^{C}t_{W}]$. We can compute P because $m = PM$, where M are the features 3D points stored before and m are the 2D projection points from the camera image, corresponding to the 3D features. In order to determine the vehicle's pose from P, we start by applying QR factorization to P, than transform from the QR to the RQ factorization and then, correct the sign of K. From this we obtain the matrices K, ${}^{C}R_{W}$ and ${}^{C}t_{W}$. Knowing that

$$ {}^{W}R_{C} = {}^{C}R_{W}^{T} \tag{15}$$

and

$$ {}^{W}t_{C} = -{}^{W}R_{C}.{}^{C}t_{W} \tag{16}$$

We extract the position from ${}^{W}t_{C}$ and the orientation angle from ${}^{W}R_{C}$.
To use this strategy, at least six features have to be detected in each iteration (see eq.6), since we have six unknown parameters. The great advantage in using this strategy, compared to the ones above, is that it allows resetting the visual odometry error, during the homing procedure. Since MonoSLAM uses EKF, exists always a probability of the filter looses its consistency, specially with abrupt rotations, meaning that estimated state and its predicted margin of error diverges from measurements. Finding known locations due to our feature map opens the possibility to reset the error in EKF prediction. A simulation using this strategy is shown in fig.14.

V. Experiments

This section describes the experiments performed to validate the methodologies introduced in previous sections.

A. MonoSLAM, self-localization error in a straight line path

After being able to run the MonoSLAM in real time, we wanted to have an idea of the pose errors. Since the MonoSLAM is very sensitive to the changes in direction, we tested it while the car was moving along a straight line.

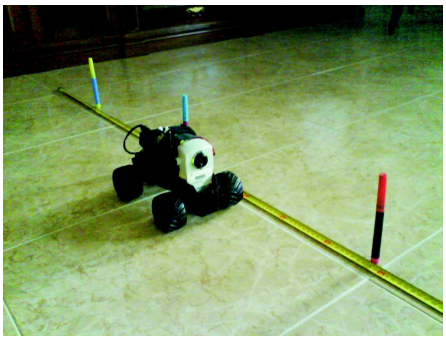
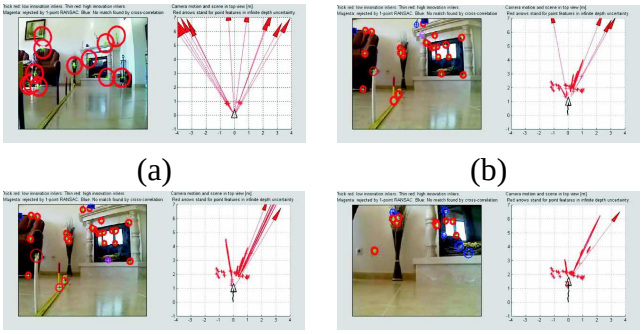


Fig. 9. Straight line experiment.



(a) (b) (c) (d)

Fig. 10. (a) First image of MonoSLAM running in the straight line experiment. (b) MonoSLAM catching a feature on the white pen. (c) Feature on white pen getting closer as the camera moves on. (d) Last image of MonoSLAM running in the straight line experiment.

MonoSLAM shows that the car is moving straight (fig.10(d)).The final orientation is similar to the real one. When a considerable number of features moves its position on the image plane from one frame to the next, the algorithm understands the movement and realizes if the camera is sliding right, left, up, down, or if it is actually moving forward or backward. After measuring the car position every 50 cm with a ruler and comparing to the position given by the MonoSLAM, we plotted the results (fig.11) to have an idea of the scale factor between the MonoSLAM direct measurements and the ruler.
After computing a linear regression of used data points, we got the line slope. By dividing the MonoSLAM measurements with the slope value, we got the measurements in the same metric scale as the ruler measurements. After plotting the scaled MonoSLAM measurements versus ruler measurements and drawing a line corresponding to the ideal case, we found how close are the MonoSLAM

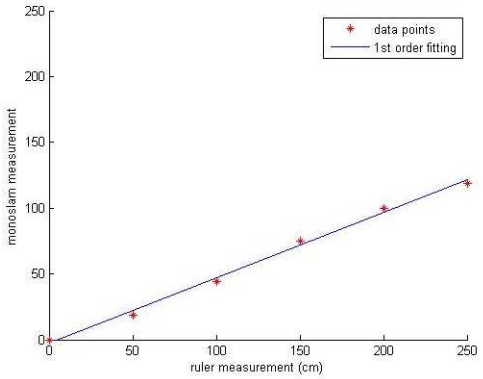


Fig. 11. Ruler vs MonoSLAM measurement to get scale factor.

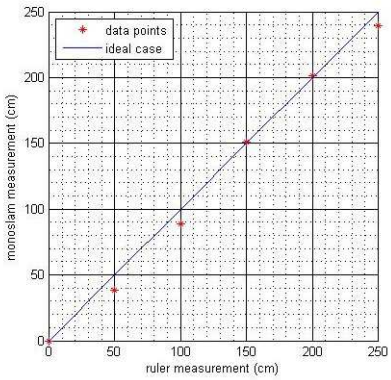


Fig. 12. Ruler vs MonoSLAM measurement in same metric scale.

measurements to the ideal case (fig.12).

B. Open Loop Homing experiment

In this experiment, we moved the robot for some time along a straight line (worst case scenario), then the homing procedure was triggered (fig.13(b)) and the robot returned to its initial position (fig.13(c) and (d)).

The difference between the MonoSLAM estimated positions and the real ones is shown in table I. The results are considerably different. The final vehicle's pose is shown in fig.13(d). From the experiments made, we conclude that MonoSLAM

	real	estimated
X (cm)	22	5
Y (cm)	-16	-8
θ (^{o})	-55	-41

TABLE I
REAL AND MONOSLAM MEASUREMENTS COMPARISON.

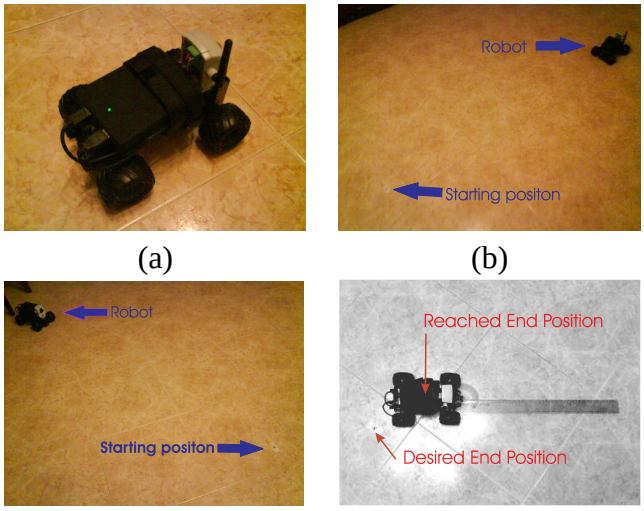


Fig. 13. (a) Start of experiment. (b) Start of homing procedure. (c) After reversing, the car goes to its initial position. (d) End position and orientation measurement.

easily detects changes in direction of motion, but its not very accurate at determining the magnitude of the forward motion of the camera, especially if there is no significant movement in more directions. The high computational cost of running MonoSLAM in MATLAB makes the job at hand difficult. No abrupt movements can be done or otherwise the difference between two consecutive images will lead to no features matches.

Using the kinematic model as the only prediction of pose while homing is possible, but leads also to a rapid increase of the pose prediction errors.

C. Map Based Homing simulation

This experiment was made to test the Map Based Homing strategy in simulation. A wide number of 3D features were spread according to normal distribution with 0 mean and unitary variance. A set of commands was sent to the simulated robot which ended up by doing a trajectory. In the meantime, the car was saving its consecutive positions in order to store a path, and a map of 3D features was being build by storing the 3D features that were in the field of view of the camera (red points in fig.14). When the robot stops receiving commands, the homing procedure is triggered and the controller takes the car back trough the inverted stored path (fig.14(b) and (c)). When the robot gets close enough to a checkpoint of the path, the reference destination

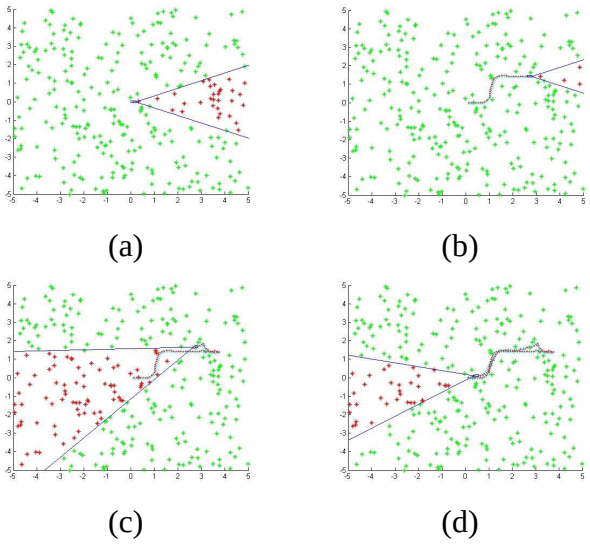


Fig. 14. (a) Beginning of Map Based Homing simulation. (b) Beginning of homing procedure. (c) Homing procedure at middle. (d) Final car pose.

becomes the next checkpoint, until the car reaches its destination. If the car can not reach a point due to restrictions in the degrees of freedom, it makes a reverse maneuver in order to get closer to the desired checkpoint. The position of the robot during this procedure is computed by decomposing the camera matrix in translation and rotation. This camera matrix is computed at every iteration using eq.6, where the 2D points (u, v) correspond to the 2D coordinates of the 3D features in the image plane, captured by the field of view of the camera.

VI. Conclusion and Future Work

This work started with the integration of the various hardware components into a mobile robot. Various positive conclusions can be extracted. The first conclusion is that wireless IP cameras are practical and functional tools to help building teleoperated robots. Also, we can say that is already possible to build a teleoperated vehicle with autonomous features at a reasonable cost. MonoSLAM was tested in a variety of experiments and, we conclude that it is possible and plausible to acquire odometry measurements using MonoSLAM but, as expected, we have considerable error accumulation. One possible way that is being considered to future use, is to decrease this error accumulation by resetting it, by comparing the actual image with the ones stored in an images map, as was proposed in Vision Based-navigation and Environmental Representations with an Omnidirectional Camera [5]. An alternative way is modifying the scenario either by adding landmarks or fixed cameras informing the car location.

Changing the features detector mechanism of MonoSLAM from corner extraction to a better one, like SIFT would improve the quality of detected features and make localization more reliable. This is an important step in order to make a Map Based Homing experiment with our setup, since this allows having more and better features. The Map Based Homing strategy is also important to reset the error associated with our EKF prediction, in case it lost its convergence.

Publications and Acknowledgments

The work described in this thesis has been partially published in RecPad 2013 [8]. This work has been partially supported by the FCT project PEst-OE / EEI / LA0009 / 2013, by the FCT project PTDC / EEACRO / 105413 / 2008 DCCAL, and by the FCT project EXPL / EEI-AUT / 1560 / 2013 ACDC.

References

[1] Jean-Yves Bouguet. Camera calibration toolbox for matlab. http://www.vision.caltech.edu/bouguetj.

[2] J. Civera, O. Grasa, A. Davison, and J. Montiel. 1-point ransac for ekf filtering. application to real-time structure from motion and visual odometry. *J. Field Robot.*, 27(5):609–631, 2010.

[3] James Crosetto, Jeremy Ellison, and Seth Schwiethale. Control an rc car from a computer over a wireless network. https://code.google.com/p/networkrccar.

[4] Mario Filipe Florêncio and Pedro Emanuel Agostinho. *Parqueamento Automóvel Automático*. Instituto Superior Técnico, 2005.

[5] J. Gaspar, N. Winters, and J. Santos-Victor. Vision based-navigation and environmental representations with an omnidirectional camera. *IEEE Transactions on Robotics and Automation*, 16:890–898, 2000.

[6] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[7] N. Leite, A. Del Bue, and J. Gaspar. Calibrating a network of cameras based on visual odometry. In *Proc. of IV Jornadas de Engenharia Electrónica e Telecomunica cões e de Computadores, Portugal*, pages pp174–179, 2008.

[8] Ricardo Ferreira Nuno Ribeiro, José Gaspar. Homing a teleoperated car using monocular slam. *RecPad*, 2013.

[9] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer, 2008.

[10] Peter Young. The leapfrog method and other symplectic algorithms for integrating newton laws of motion. http://young.physics.ucsc.edu/115/leapfrog.pdf, 2013.