Universidad Autónoma de Madrid

Escuela Politécnica Superior

Double Degree in Computer Engineering and Mathematics

TRABAJO DE FIN DE GRADO

BACHELOR'S THESIS

CONTROL VISUAL DE UN ROBOT MÓVIL MEDIANTE UNA CÁMARA CENITAL

VISUAL CONTROL OF A MOBILE ROBOT BY MEANS OF AN OVERHEAD VIEW

Iván Márquez Pardo
Tutor: Luis Fernando Lago Fernández

June 2016

VISUAL CONTROL OF A MOBILE ROBOT BY MEANS OF AN OVERHEAD VIEW

Autor: Iván Márquez Pardo
Tutor: Luis Fernando Lago Fernández

Departamento de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

June 2016

Acknowledgements

This project is the milestone that represents the successful completion of my *Double Degree in Computer Engineering and Mathematics* at *Universidad Autónoma de Madrid*.

These five years as a student at *Escuela Politécnica Superior* have made a huge impact in my life and I think the experience acquired through these years has helped me grow as a person.

I wish to express my sincere gratitude to all the teachers of the university, but specially to Luis Fernando Lago Fernández, who has been my tutor in this thesis. His constant support, the interest he has always demonstrated and his willingness to help me in every moment have been a huge encouragement to finish this thesis. I cannot thank you enough!

I would like to thank my classmates, without them it would have been impossible for me to finish this degree. They have always encouraged and helped me during these 5 years; they are the best companions anyone could have ever imagined. I also thank the *Club de Robótica y Mecatrónica* of *Universidad Autónoma de Madrid*, whose members have helped me when I had no idea of programming an Arduino script.

Finally, I want to express deeply gratitude to my family and friends. They have always been comprehensive and supportive in regards to my studies, not complaining because of my lack of spare time during the course. They are the pillar that holds me day by day.

"We learn little from victory, much from defeat." Japanese Proverb

Abstract

Abstract —

This research project addresses the problem of controlling the motion of a small mobile robot by means of visual feedback provided by an overhead camera. This visual servoing problem has been previously addressed by many researchers due to its multiple applications to real world problems. In this document, we propose a software platform that rely on low cost hardware components to solve it. Based on the imagery supplied by the overhead camera, the proposed system is capable of precisely locating and tracking the robot within a planar ground workspace, using the CAMShift algorithm, as well as finding out its orientation at every moment. Then, an error measurement is defined between current and desired positions of the robot in the Cartesian plane (Position-Based Visual Servoing). In order to generate the suitable motion commands that lead the robot towards its destination, we make use of mathematical equations that model the control of the robot. The platform has been especially designed regarding its application to real time problems.

One of the central goals of this work is analyzing the viability of the proposed system and the level of accuracy that it is capable of achieving taking into account the low cost components on which it is based. The validation of the system has come as a result of the real time experiments that have been conducted. Firstly, an exhaustive battery testing that comprehends 1400 experiments has been conducted in order to find a suitable set of parameter values that polished the control equations. Secondly, we have implemented three different applications to test these new control values: tracing a trajectory defined by a fixed set of points, pursuing a mobile target and integrating our system with a block-programming platform from which it receives a set of destination points to be followed. Having successfully completed all these tasks, we conclude that the proposed robotic system has well proven its feasibility and effectiveness facing the addressed visual servoing problem.

Key words — Software platform, Python, OpenCV, computer vision, perspective, distortion, location, orientation, Arduino, Bluetooth, servo, visual servoing.

Resumen

Resumen — Este proyecto de investigación aborda el problema de controlar el movimiento de un pequeño robot móvil por medio del feedback visual proporcionado por una cámara cenital. Este problema de control visual de servos ya ha sido abordado previamente por multitud de investigadores debido a sus múltiples aplicaciones a problemas del mundo real. En este documento, se propone una plataforma software que depende de componentes hardware de bajo coste para resolverlo. Basado en imágenes suministradas por la cámara cenital, el sistema propuesto es capaz de localizar y seguir de forma precisa al robot dentro de un entorno de trabajo en el plano del suelo, usando para ello el algoritmo de tracking CAMShift, así como averiguar su orientación en cada momento. Después, una medida de error se define entre la posición actual del robot y la deseada en el plano Cartesiano (control visual de servos basado en posición (PBVS)). Para generar los comandos de movimiento aporpiados que lleven al robot a su destino, hacemos uso de ecuaciones matemáticas que modelizan el control del robot. La plataforma ha sido especialmente diseñada teniendo en cuenta su aplicación a problemas en tiempo real.

Uno de los objectivos centrales de este trabajo es analizar la viabilidad del sistema propuesto y el nivel de precisión que es capaz de obtener teniendo en cuenta los componentes de bajo coste en los que se basa. La validación del sistema viene dada como resultado de los experimentos en tiempo real que se han llevado a cabo. Primeramente, una exhaustiva batería de pruebas que comprende 1400 experimentos ha sido ejecutada con el fin de obtener un set de valores para los parámetros que puliesen las ecuaciones de control. A continuación, hemos implementado tres aplicaciones diferentes para probar estos nuevos valores de control: trazar una trayectoria definida por un conjunto de puntos fijos, perseguir un objetivo móvil e integrar nuestro sistema con la plataforma de programación por bloques desde la que recibe el conjunto de puntos a seguir. Habiendo completado todas estas tareas satisfactoriamente, concluimos que el sistema robótico propuesto ha demostrado con holgura su viabilidad y efectividad frente al problema de control visual de servos abordado.

Palabras clave — Plataforma informática, Python, OpenCV, visión por ordenador, perspectiva, distorsión, localización, algoritmos de seguimiento, orientación, Arduino, Bluetooth, servo, control visual de servos.

Glossary

Arduino A family of open-source boards generally designed around a single 8-bit Atmel AVR microcontroller. Current models feature a USB interface, analog and digital I/O pins which allows the user to attach various extension boards. (http://www.arduino.cc/). 14, 16, 18, 23

Bluetooth Created in 1994, Bluetooth technology was conceived as a wireless alternative to data cables by exchanging data using radio transmissions. It is the global wireless standard enabling the Internet of Things (IoT). Bluetooth not only connects devices together in an ultra-power efficient way, but also directly connects devices to applications on smartphones, PCs or tablets. (https://www.bluetooth.com/). 14, 16, 18, 23, 24

Features In computer vision context, they refer to detectable parts of an image which can be used to support a vision task. 9

OpenCV OpenCV, which stands for Open Source Computer Vision, is a computer vision library free for both academic and commercial use, which has multiple interfaces (C++, C, Python and Java) and supports multiple operative systems (Windows, Linux, Mac OS, iOS and Android). OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. (http://opencv.org/). 13, 18–20

PrintBot PrintBots, a.k.a. PRINTable roBOTS, is a family of robots that are open-source and can be manufactured using a low-cost 3D printer. PrintBots are oriented to the community: people in different countries can download and print each of the parts that make a PrintBot, and also modify them and re-share the improvements with the rest of the world.. 17

Python A widely used general-purpose, high-level programming language that is cross-platform. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than what would be possible in other low level languages such as C. (https://www.python.org/). 13, 14, 18

Acronyms

ABCShift Adaptive Background CAMShift. 11

CAMShift Continuously Adaptive Mean Shift. 11, 13, 21–23, 33

FOV Field Of View.. 7

FPS Frames Per Second. 18

IBVS Image-Based Visual Servoing. 7, 8

ORB Oriented FAST and Rotated BRIEF. 10, 21

PBVS Position/Pose-Based Visual Servoing. 7, 8, 11

PoE Power over Ethernet. 18

Contents

List of Tables

List of Figures

1 Introduction

The aim of this project is the development of a software platform that is capable of generating a trajectory and controlling a robot along it just by means of an overhead view provided by an IP camera.

1.1 Motivation

Controlling the motion of a robot by means of an overhead camera is an important case of the visual servoing problem that has captured the interest of many researchers due to its potential applications to real world situations.

The idea for this project came up as the result of our desire of making robots programming achievable for preschool children. The whole idea consists in developing an educational environment that allows the programming of the robot movement using blocks. Each of these blocks has a directional arrow painted on it indicating the desired movement of the robot, so that the child is capable of defining a trajectory for the robot to follow just by joining the suitable combination of blocks. Once the trajectory is defined, the robot will try to follow it within its defined workspace, based on the visual feedback provided by an overhead camera that films the scene. This is where our project will be necessary: we need a trusty method to control the trajectory described by that robot.

In [1], Irene Colmenar has developed a web application called ARDUAMBOT, directed towards young learners that is able to control the motion of a robot by means of graphical programming blocks. These blocks represent basic moves (four cardinal directions) as well as more complex rules (loops and conditionals) that the robot is expected to be able to perform.

Typically, this application proposes a problem to the user in the form of a certain path that must be followed by the robot in the real world and the child has to define, aligning the suitable blocks, the specific moves that will make the robot complete that task. Then, the web application turns those blocks into a set of spatial coordinates and sends them to our platform. Finally, the visual servoing platform takes control of the robot, sending it the corresponding motion orders to achieve completion of the specified trajectory.

This clearly demonstrates that the system we propose in this document is valid and applicable to solve a real world visual servoing problem.

1.2 Scope

This project addresses the problem of controlling the motion of a robot in real time using the visual feedback provided by an overhead camera. The main purpose of this control system is being part of a block-programming software platform specially designed as an introduction for preschool children to robotics. Thus, it has been developed emphasizing its modularity, adaptability and as a low cost solution to our problem.

Our approach to the desired solution is based on a computer that receives visual feedback from a single IP camera and sends the corresponding movement orders to one robot, placed inside a planar workspace that is being recorded by that camera. These motion orders are sent wirelessly using Bluetooth technology from the computer to the Arduino board that controls the robot, which has integrated its own Bluetooth module.

As we have been working with low cost components and hardware that were given, we exclude from the scope of this project studying their level of suitability or if there were any better options. In this sense, this project does not address, for example, the problem of choosing the most effective wireless technology for sending orders to the robot (i.e. Bluetooth vs. Wifi), nor the quality of the servos or the Arduino board that are part of the assigned robot, nor the transmission speed difference between our IP camera and an USB one.

Therefore, this document proposes a low-budget platform that solves the visual servo control problem of controlling the motion of a vehicle moving within a limited workspace based on visual feedback acquired by an overhead camera. Experimental results have validated the feasibility of the proposed platform, thus encouraging further developments or improvements in this direction.

1.3 Project approach and methodology

There are several requirements that must be met prior to developing the control of the robot, such as calculating the perspective correction matrix that produces the desired overhead view and being able to precisely locate the robot as well as its current orientation within the established workspace.

Then, after achieving a rough and basic robot control, we have tried to polish it adjusting the parameters of the mathematical equations which describe the motion of the robot. For this purpose, we have designed a battery testing to find the most suitable parameters and then tested them with different applications: following a fixed trajectory and pursuing a mobile object.

Finally, as a last application, we have managed to integrate our system with the block-programming platform described in [1].

Given the list of goals to achieve with this project, we have followed a combination between both top-down and bottom-up approaches. In this sense, firstly the main modules that compound our desired system have been determined, defining their complete functionalities and how they would interconnect with each other once implemented. Then, an agile methodology has been applied to develop functional modules that were tested before their integration with the rest of the software system.

1.4 Structure of this document

This document is organized as follows. In Section 2, we review relevant work on the visual servoing problem, focusing on tracking and control techniques applied in the matter. Section 3 summarizes the system's design, whose most important implementation issues are detailed in Section 4. Section 5 presents the results we have obtained on experimental testing and three different applications of our system which validate its feasibility. Finally, Section 6 concludes and proposes further enhancements of the system.

2

State of the Art

In the past decades, technology has become more and more accessible and affordable to the public. In this sense, due to the widespread availability of high quality cameras and low cost microprocessors, the use of computer vision techniques to control robotic systems has gained in popularity [2]. This field of study is called visual servoing and it has awaken the interest of many researchers who can see its enormous potential.

Many of the applications of visual servoing refer to the control of an end-effector robot using visual feedback with the aim of making it follow a defined trajectory or move to a set point in the real space. There is a lot of previous work in this area due to the wide range of applications; as an example, [3] proposes a model to control align a robot with parking lot lines, [4] and [5] address the problem of controlling the motion of a robot within a warehouse and [6] models an autonomous robot capable of navigating in unknown and dynamic environment to seek a target without colliding with obstacles, suitable for disaster monitoring or security patrol.

Other more complex applications include robot-assisted surgical procedures like in [7], [8] and [9], autonomous space robots capable of capturing faulty satellites [10] or the implementation of distributed and cooperative strategies like those used in the RoboCup to achieve, for example, that a group of five robots play a football match coordinating as a team [11] [12].

2.1 Visual Servoing

Vision-based robot control, also known as visual servoing, is the field of robotic vision that uses visual feedback provided by a camera in order to control the motion of a robot.

Visual feedback is the fusion of results from many elemental areas including high-speed image processing, kinematics, dynamics, control theory, and real-time computing [13].

Providing robotic systems with vision enhances their ability to make real-time decisions based on that imagery, which in addition to information from other sensors (e.g. ultrasound, infrared, accelerometer, gyroscope, etc.) will allow them to perform much more complex tasks [9].

Two fundamental configurations can be distinguished in the application of visual servoing, both having its own merit and drawbacks regarding its precision and sight range:

- Eye-in-hand: In this technique, the camera is attached to the moving robot, observing the relative position of the target. The rigidly mounted camera has a partial but precise sight of the scene but there is no possibility to interact with the whole workspace because of the narrow field of view. In comparison with an static and fixed camera, this architecture is far more versatile in terms of accuracy during inspection and assembly, thus being suitable for robot arms in manufacturing factories [14].
- Eye-to-hand: This technique comprehends a fixed camera, observing the robot and its motion. This configuration has a less precise but global sight of end-effector robot within its workspace. With an overall view of all the scenery it is possible to calculate optimal trajectories beforehand, thus being appropriate for industrial manipulators as they would be following precomputed collision-free and occlusion-free paths [15].

In addition to these fundamental configurations, it is worth mentioning the existence of hybrid solutions that combine both of them in order to perform complex tasks, taking advantage of their complementary strong points whilst resolving their individual problems [16]. Despite the increased complexity of control methods in hybrid configurations (mainly caused by the need of coordination between eye-in-hand and eye-to-hand imagery), experimental results show an improvement in precision and effectiveness whilst completing tasks in comparison with simple configurations [17]. In [17] and [18], eye-to-hand approach is considered for target pose (position and orientation) and velocity estimation followed by a more accurate pose detection and tracking of the target achieved by an eye-in-hand approach.

2.2 Control Techniques

Given a visual servoing problem, it is necessary to choose a suitable control technique depending on the characteristics/configuration of the problem. The basis for control techniques is the definition of an error measure between the current and the desired state of the system to generate a motion command whose purpose is minimizing that error in the next iteration, thus leading the system to the desired state.

In the context of visual servoing, these control methods can be broadly classified into the following types [13]:

- Position/Pose-Based Visual Servoing (PBVS): In this method, significant features are extracted from the image and used to estimate the pose (position and orientation) of the target with respect to the camera. Then, using these values, an error between the current and the desired pose is defined in the task space.

  In PBVS, the vision sensor is considered as a 3D sensor since the actuator and the trajectory are expressed in the Cartesian space. A correct estimation of the pose is crucial in this method, as it affects the error computation; coarse estimations thus induce perturbations on the trajectory followed, having a significant impact on the accuracy of the pose reached [19].

  Using an eye-to-hand configuration, [20] developed an algorithm based upon PBVS to provide a collision-free global path planning for a robot.

- Image-Based Visual Servoing (IBVS): In this technique, the error signal is defined directly in terms of image feature parameters, in complete opposition to position-based methods that define the error in the task space coordinates.

  In IBVS, the vision sensor is considered as a 2D sensor since the features are directly expressed in the image space, making it remarkably robust to errors and image noise. As for the 3D parameters involved, a poor estimation can make the system unstable, but coarse ones will only imply perturbations in the trajectory followed by the robot to reach the desired pose, not on the accuracy of the pose reached [19].

  Again, with an eye-to-hand configuration and IBVS, [4] proposes a feasible method for generating trajectories that can be followed by a robot thanks to an in-depth study of the differential flatness of image based systems.

- Hybrid Approach: Depending on the proposed visual servoing problem, it might be necessary to implement a hybrid approach in order to solve it with a certain level of precision that otherwise would be unreachable for PBVS or IBVS alone.

  In general, hybrid approaches are obtained as a combination of IBVS and PBVS (e.g. IBVS for translation; PBVS for rotation), by switching them based on some performance criteria (e.g. IBVS when feature is about to leave Field Of View. (FOV); PBVS otherwise) or as an addition of both (e.g. IBVS maintains FOV constraint, PBVS guarantees global stability and performance) [21].

  With an eye-in-hand configuration, a hybrid 2D/3D visual servoing is applied in [10] to switch between IBVS and PBVS when the target is near enough to capture it with a robotic arm. In this scheme, IBVS is used for translating the robotic arm due to its simplicity and low sensitivity to camera calibration errors whilst PBVS, which has a higher sensitivity to that kind of errors and requires an accurate 3D model of the target, is only applied near enough for rotating the end-effector arm.

2.3 Tracking Algorithms

As previously mentioned, servo control systems need the current state of the end-effector robot in order to generate a suitable motion command that leads the target towards its destination. Therefore, being capable of tracking the target through several frames is a basic requirement for solving visual servoing problems.

Video tracking is the problem of "following image elements moving across a video sequence automatically" [22]. Tracking systems must address two basic problems: motion and matching.

The motion problem consists of predicting the location of the tracked image element in the next frame. In other words, it seeks identifying a limited search region in the image in which the element is expected to be found with a high probability. The simplest approach to this problem is to define the search area for the next frame as a fixed-size window around the target position in the previous frame; the size of this window is dependent on the problem and the expected frame-to-frame displacement, thus performance is limited. Here, the Kalman filter enters the scene as a well-known solution to this problem [22]. The Kalman filter [23] is an optimal, recursive estimator of the state of a linear dynamic system, which firstly makes an educated guess about the evolution of that system and then uses uncertain information (usually provided by noisy sensors) to correct that prediction, improving its accuracy. Kalman filters have been adopted widely in video tracking since they are light on memory and can be run in real time [24], achieving an efficient tracking of multiple objects under confusing situations [25].

The matching problem aims at identifying the image element in the next frame within the designated search region. In order to detect the target, it requires a similarity metric to compare candidate pairs of image elements in the previous and current frame. Choosing the most suitable tracking algorithm for each problem is crucial since some of them can only follow a single target and may get lost if other candidate enters the search region whereas others are capable of following multiple targets with the drawback of a lower performance due to extra image processing [22].

Among the requirements for an ideal video tracker, there are robustness to clutter (do not get distracted by other candidates), robustness to occlusion (if the target is temporarily occluded, resume tracking when it reappears), false positives/negatives (ignore everything apart from valid targets), agility (do not lose the target despite changes in its speed and acceleration) and stability (tracking accuracy must be maintained over time).

There are some fundamental video tracking techniques developed by the computer vision community. Whilst the motion problem is nearly invariably solved (simple predictions of the search region are enough in most cases), algorithms for solving the matching problem heavily depend on the complexity of the target. According to [22], these techniques can be classified as follows:

- Window Tracking: the simplest target is just a small image window. In other

words, in the next frame we will look for a rectangular region that matches the small window we took as a reference from the first frame (also known as the correspondence problem, studied in [26]). The search region in the next frame can be determined by a Kalman filter or simply centering a fixed-size window around the previous target position.

- Feature Tracking: firstly, some relevant image Features are identified in the current frame, then these features are extracted from the next frame and a correspondence is established between those two sets. This way, the motion of the target is estimated as a difference between particular features in subsequent frames (not just comparing entire windows like in window matching). Depending on their complexity, Features can either be local (e.g. edges, color, corners or lines) or extended (e.g. regions or contours). Feature tracking has awaken the interest of many researchers; for example, [27] proposes an algorithm that compares edge segments from successive frames, being capable of detecting motion and tracking multiple targets with the aid of a Kalman filter.

- Planar Rigid Shapes: the target is a planar rigid object whose properties are known beforehand. Its shape with respect to the camera along with an imaging projection gives information about target position and orientation. Shape-based pattern matching over QR-codes is used in [28] to guide human-tracking robots.

- Solid Rigid Objects: in this case, the target is a solid rigid object whose model is available and thus it is capable of providing 3D position and orientation over time. Tracking and recognition of predefined hand gestures is achieved in [29] based on a mathematical model of a hand.

- Deformable Contours: this tracking method is based on a model with a limited number of parameters that predict changes in shape and motion of a deformable object. An example of tracking objects whose contours change over the time can be seen in [30].

- Visual Learning: it incorporates machine learning to computer vision, which enhances performance and multiplies the abilities of this field. The system automatically learns the shape and dynamics of complex targets from lots of images or videos without any a priori information nor model of the target. An analysis in the use of neural networks to automatically detect good features to track is performed in [31].

Since each tracking technique has its advantages and its drawbacks, research has been widely conducted in order to study hybrid solutions that apply two or more tracking algorithms to the same target. The work in [32] proposes a framework for combining tracking algorithms: firstly, two trackers are manually initialized to follow different features of the target and then, independent Kalman filters propagate those states, thereby the final tracking result is obtained as an integration of the partial results from those filters. In [33], multiple windows run the Mean Shift [34, 35] algorithm over different colors, updating the model of the target over time. This model update results in an

algorithm that outperforms the original Mean Shift thanks to its enhanced robustness in the presence of large varying features.

Other hybrid solutions are oriented to the combination of different tracking algorithms so that one overcomes the deficiencies of the other and vice versa. In this sense, both [24] and [36] suggest a Mean Shift algorithm based on Kalman filter. This approach firstly uses the Kalman filter to predict the target location and then uses that point as a seed for the Mean Shift algorithm that searches the target around it. This way, the complete algorithm is real-time computationally feasible and achieves more accurate tracking overcoming the main drawbacks of the original Mean Shift (e.g. it is more stable in the presence of fast moving targets and, to a certain extent, it avoids losing target when object and background colors are similar). In [37], an improved Mean Shift tracking based on Oriented FAST and Rotated BRIEF (ORB) [38] corner feature detection is presented. First of all, the target area is selected and the color histogram is calculated for Mean Shift, whereas a model of the target is extracted for ORB; during the tracking, if Mean Shift is unable to estimate the target position to some extent, object location is given by ORB feature matching based on the acquired target model. Therefore, this method enhances object location accuracy, suppressing the critical drawbacks of the former Mean Shift.

2.3.1 Mean Shift Algorithm

The Mean Shift algorithm [34, 35] has been previously mentioned whilst describing hybrid tracking solutions, but it is worth explaining its properties, as it has played a fundamental role in this project.

Mean Shift belongs to feature-based tracking techniques, as it uses a color histogram of the target in order to locate it in the image. Pixels that lie within the search region (fix-sized rectangular window) are given a probability that they belong to the tracked object, using target's color histogram for this purpose. This way, a distribution of object location has been created over a local area of the image. The tracking problem is solved by mean shifting towards the centroid of this distribution, estimating it as the new location of the target. The search window is centered in this new position and the process is iterated until convergence. This algorithm has achieved considerable success solving tracking problems due to its simplicity, robustness against target partial deformation and computational efficiency, thus being real time applicable [37]. However, tracking small or fast moving targets is prone to failure and self-recovery track is not guaranteed [24].

Continuously Adaptive Mean Shift (CAMShift) [39] is an enhancement of the earlier Mean Shift algorithm in which, instead of the fixed-size tracking window, an adaptive window is used. This search region resizes depending on the size of the target thus adapting itself consequently whenever the object gets closer or moves away from the camera. Researchers have already proposed better options than standard CAMShift: Adaptive Background CAMShift (ABCShift), introduced in [40], improves CAMShift robustness against loss of target due to background changes, whereas modifications of standard CAMShift in [41], help dealing with occlusion and illumination problems.

2.4 About this project

This project belongs to the visual servoing field, as its main purpose is tracking and controlling the motion of a robot in real time using visual feedback provided by an overhead camera.

The experiments have been conducted using an eye-to-hand configuration with imagery provided by a fixed camera placed over the workspace. Similarly to [4], we have used a single camera that has been calibrated prior to experimental tests, although future work in this project takes into account the extension to multiple (perhaps uncalibrated) cameras with the aim of expanding the image space and therefore being able to control the robot in a larger workspace.

The control technique that has been chosen for this project is PBVS. Even though the tracking is performed in the original image, the coordinates are transformed with a perspective correction, thus estimations of position and orientation are computed in the Cartesian plane of the transformed image. It is assumed that the robot moves within a ground planar workspace; this fact justifies that we have worked with coordinates in the Cartesian plane, not in the real 3D space.

With regard to the tracking algorithm, CAMShift has been applied for this project. Despite the standard CAMShift may lack enough robustness facing certain situations (see 2.3.1), the decision was made based on previous successful experience in working with it (Carlos García applied it in his Bachelor's Thesis [42]) and it has indeed produced good results when applied to this project (in any case, we leave the door open to enhance the tracking algorithm in the future).

3

System Design

This chapter aims to introduce a general idea on the design of the system and the subsystems in which it has been split after analyzing its functional requirements.

3.1 Subsystems

The software platform has been designed following a top-down approach to define its main modules, their expected functionality and their intercommunication interfaces. The subsystems determined during this phase of the development were later implemented as Python classes in order to ensure the modularity of the system. Sorted by order of complexity (each module relies on the previous one), the system has been split into these subsystems:

- Camera: The main purpose of this class is capturing the images of the workspace provided by the IP camera. Furthermore, it is in charge of the camera calibration (removal of lens distortion enhances the accuracy of captured frames) and the perspective correction (a homography is applied to the original image in order to make up for the camera not being placed directly over the workspace). Both the calibration and the perspective correction are automatic processes whose resulting parameters are stored in configuration files, thus being necessary to run them only once (each time the camera changes its position).

- Tracker: This class performs the tracking of the target relying on the visual feedback provided by the Camera class. Since the tracking algorithm, CAMShift, is already implemented in the OpenCV library, those functions are conveniently wrapped to

let the user select the specific area to track in the image. Apart from the location of the target, this module implements a method to obtain its orientation. Moreover, the tracker is responsible for sending the current pose (position and orientation) of the object to the Robot Controller.

- Robot Controller: This module is highly dependent on the tracker, as it needs a constant update of the current pose of the robot in order to properly work. Given a trajectory and using robot's current position and orientation provided by the tracker as feedback, the controller is in charge of generating and sending to the robot the suitable motion commands (via Bluetooth) to make it follow that path. In the particular case of the integration with Arduambot, the block-programming platform of [1], this class represents the point of connection from which the desired trajectory is obtained.
- Launcher: This class is a simple wrapper of both the tracker and the robot controller classes. It is in charge of running both the tracker and the robot controller in different processes, choosing their appropriate operating modes based on a configuration file and establishing a pipe as the intercommunication channel between them.
- Arduino Board Controller: Apart from these Python classes, the design of the system includes an Arduino program for controlling the robot. It continuously reads the motion commands sent via Bluetooth by the robot controller and causes the effective move of the servos attached to the Arduino board.

It is worth mentioning the modularity and expandability achieved with this design: it was manageable to add new functional requirements that appeared during development and modify existing ones with minimal changes in the affected module without it interfering with the others.

3.2 System's scheme

In order to make the robot reach a particular position in the workspace, the required information flows within the system following the scheme represented in figure 3.1:

0. If the block-programming platform is connected to the system, it sends the desired set of points to the robot controller; otherwise, the path to follow is generated by the controller module itself. Once the trajectory has been determined, the tracker receives a message from the controller indicating that it must start sending current target's pose.
1. The camera subsystem captures a frame which represents the current state of the end-effector robot within the workspace.
2. The tracker receives the transformed frame from the camera module, processes it and extracts current pose (position and orientation of the robot).

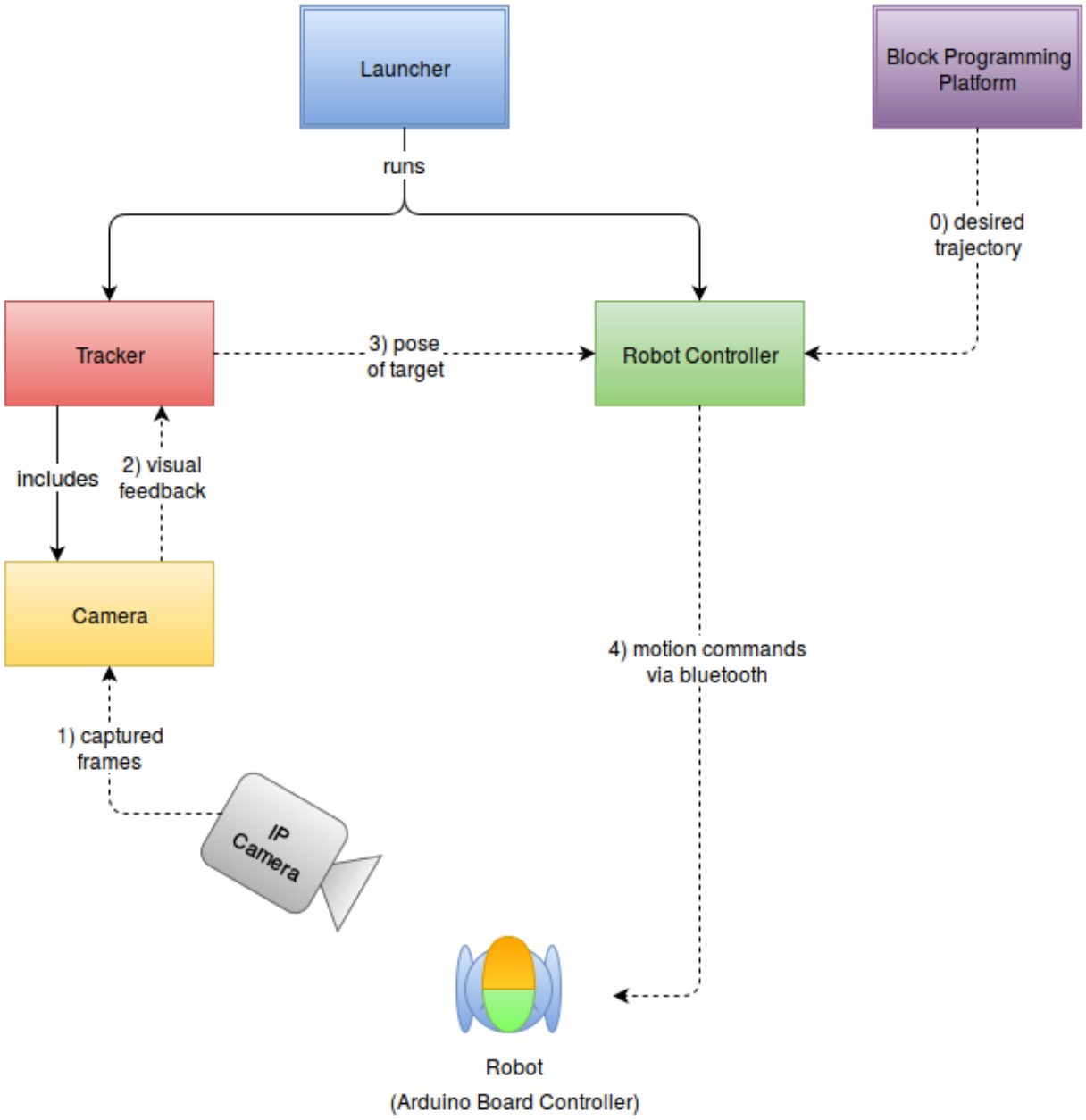


Figure 3.1: Block diagram of the proposed system architecture. Continuous lines indicate relationships whilst discontinuous lines represent information flows between subsystems.

3. Target's current pose along with its timestamp is sent to the robot controller module.

4. At this point, if the robot has already reached its destination, the controller sends a message to the tracker to stop its information flow and then it can either enter in standby mode waiting for another trajectory (if the block-programming platform is connected) or make the whole program finish (in case of an auto-generated path). Otherwise, the robot controller uses up-to-date information about robot's status in order to generate a motion command that corrects its movement leading it towards the desired destination.

5. The Arduino board controller receives via Bluetooth the motion commands generated by the robot controller and forwards them to the servos, causing an effective movement of the robot.

6. Current state of the target has changed. Return to step 1.

4

System development

Once the subsystems were designed, they were implemented using an agile methodology, following a bottom-up approach. The choice of this methodology allowed the development of independent modules whose functionalities were fully tested before their integration with other modules; this way, debugging was usually limited to the newest module that had been integrated.

This chapter provides an overview of the main aspects of the system's development, describing how they were implemented and justifying some relevant decisions that were made during this phase.

4.1 Hardware

First of all, it is necessary to describe the hardware components we have worked with in order to develop the system, since they played a fundamental role in the decision making.

The robot was a PrintBot lent by the research group *Grupo de Neurocomputación Biológica* at *Universidad Autónoma de Madrid*. In particular, the design of this robot is work of Carlos García-Saura, former student of this university, in whose Bachelor's Thesis designed a robot of similar specifications to this one. For further information about the GNBot project, visit https://github.com/carlosgs/GNBot/. The use of PrintBots is quite convenient since they provide an easy way of designing robots that are suitable for research, allowing the reuse of previous designs and the incorporation of new components needed to perform a certain task [42].

The robot consists of a printed structure that holds two SpringRC SM-S4303R servos (datasheet supplied in Appendix A.1) and a BQ ZUM BT-328 board (figure 4.1).

Fabricated by BQ, this board is 100% compatible with Arduino and has its own Bluetooth module integrated. Thus, this module was the reason to choose Bluetooth technology over other wireless solutions (e.g. Wifi) in order to solve the communication problem between the robot and our system. Technical specifications and schematics of the ZUM BT-328 board are supplied in Appendix A.2.

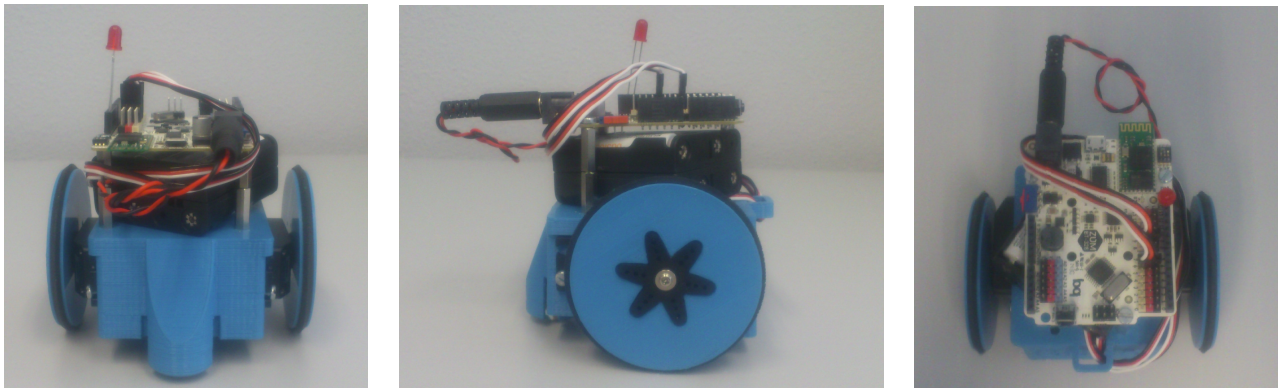


Figure 4.1: Close-up of the robot. The BQ ZUM board is powered by a set of 8 AAA batteries and has two SpringRC SM-S4303R servos and a red LED attached to it.

With respect to the overhead camera (figure 4.2), we have been given a D-Link DCS-3110 Megapixel Power over Ethernet (PoE) Network Camera. This surveillance camera was configured to be remotely accessible from its own IP, thus the system was specifically designed to access that IP in order to read video frames. The camera was set up to provide images up to 30 Frames Per Second (FPS) at a resolution of 640x480 pixels. The reason to use this setup instead of working at 1280x1024 pixels is that this higher resolution mode has a limited framerate of 8 FPS and besides, the desired real time performance is badly affected when trying to process these high-res images. Further information about camera specifications is included in Appendix A.3.

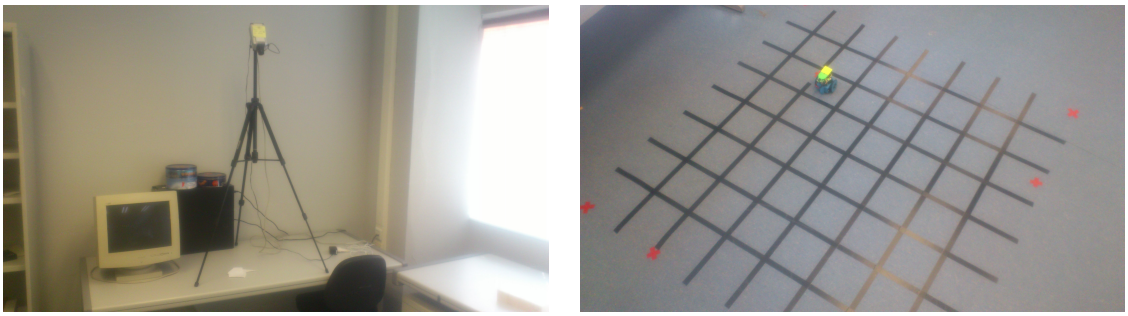


Figure 4.2: Detail of camera installation and workspace. The camera is attached to a tripod so that it provides an overhead view of the workspace. The workspace presents a grid, which is specially useful in the particular case of the integration with the block-programming platform.

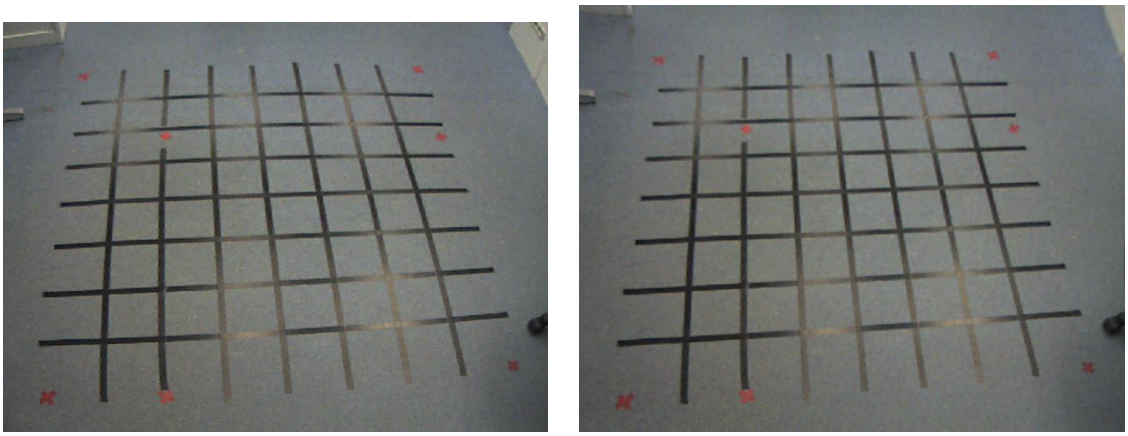
Finally, our system was implemented in Python 2.7 and OpenCV 3.0 and was tested in Ubuntu 14.04, x64 bits, running on a computer with an Intel Core 2 Duo CPU E8500 @ 3.16GHz x 2 processor and 4.0 GB of RAM.

4.2 Image preprocessing

Prior to performing the robot tracking, the images captured from the camera go through a preprocessing: calibration and perspective correction.

4.2.1 Camera calibration

The objective of camera calibration is removing the camera's lens distortion from the image, improving its fidelity and thus enhancing the future tracking process. Distortion is a deviation from the rectilinear projection and although the camera we have used does not produce such a perceptible deviation (mean reprojection error in this case was 2.27736542979 pixels), in general it is considered to be a good practice to calibrate the camera before working with it. In the future, changing the current camera for another with higher lens distortion (e.g. fisheye cameras), will not be a problem, as the calibration is already implemented. The result of the calibration process can be seen in figure 4.3



Original capture Undistorted capture

Figure 4.3: Result of the camera calibration. The removal of the camera distortion in the original capture on the left produces an output on the right in which the lines of grid pattern of the workspace are straighter.

The camera calibration that has been implemented is based on detecting the inner corners of a 10x7 chessboard (thus, 9x6 corners) from different angles, positions and distances to the camera. OpenCV then takes the detected corners and, knowing that they belong to a chessboard pattern, applies the suitable image corrections to properly align them. These image transformations undistort the original image. A sample of the images used in the chessboard pattern recognition appears in figure 4.4.

4.2.2 Perspective correction

The next step is correcting the perspective of the undistorted image. This process aims to compensate the fact that the camera is not strictly vertically aligned over the workspace,

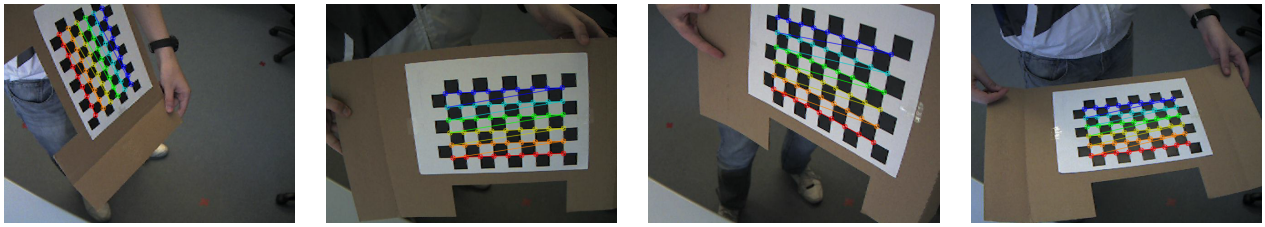


Figure 4.4: Sample of chessboard pattern recognition. A total of 25 photos with the chessboard in different positions were taken in order to achieve camera calibration.

but simply mounted overhead.

The image perspective is corrected by means of a transformation from four points (coordinates) of quadrangle vertices in the source image to coordinates of the corresponding quadrangle vertices in the destination image. We defined a ground planar squared workspace limited by red landmarks at the corners; the centre of these four marks determine the four source points. The four destination points are manually defined to form a square. This way, the transformation we are looking for is such that the image is deformed to in order to translate the source points to their corresponding destination coordinates. OpenCV defines this correction perspective in terms of a matrix that, applied to the whole original image, produces a simulated bird's-eye view of the workspace. The final result of the perspective correction can be seen in figure 4.5

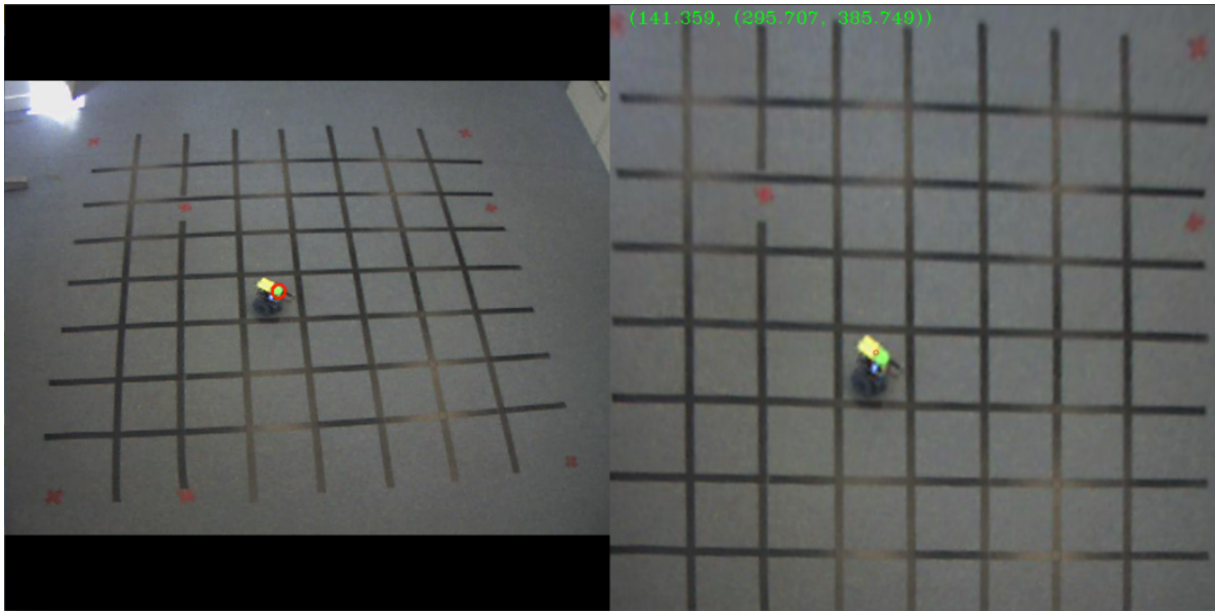


Figure 4.5: Final result of the perspective correction. The transformed image provides a more accurate view of the real world, as if the camera was above the workspace.

A matter of implementation is how the source image points are obtained in an automatic way: a mask is applied over the original undistorted image in order to filter the red colour and then, the centres of the four biggest red contours detected are extracted, each one of those representing a source point. Figure 4.6 represents the result of this

procedure.

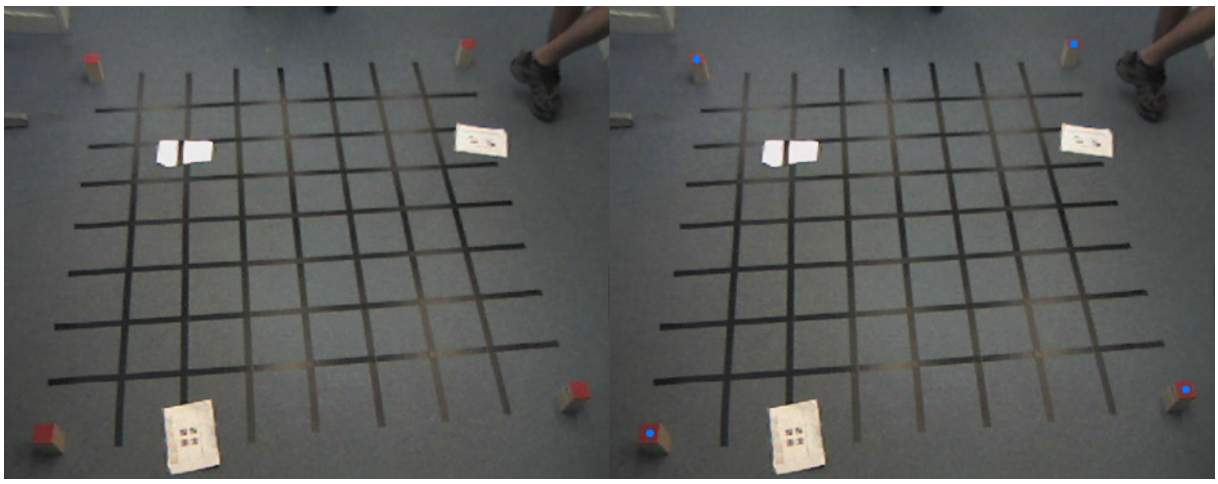


Figure 4.6: Source image reference points extraction. Landmark coordinates are extracted applying a red mask over the original image.

4.3 Tracking algorithm

CAMShift is the tracking algorithm we have finally applied to our visual servoing problem. Nevertheless, it is not the only option we considered during the system's development. As a matter of fact, we knew that CAMShift worked and it was applicable to this problem given the previous work of Carlos García-Saura with the algorithm [42], but we did not want to resign ourselves to applying it without taking into account other options.

The first attempt consisted in developing an hybrid solution, improving standard CAMShift colour-based tracker with ORB key points feature tracker (similarly to the one in [37]). This naïve approach was discarded when we noticed that the low resolution (640x480 px) of the image was an obstacle to key points detection and besides, the key points model of the robot changed every time the robot moved or span, thus being necessary to constantly update ORB's object model (final result was worse than the original).

Later, we tried to apply template matching to achieve robot localization. Choosing QR codes as templates could give us information about robot localization and orientation, all in one. This option was extremely scalable, as several robots could be uniquely identified just by looking at their corresponding QR codes. Again, this option presented a problem with the low resolution of the image, which prevents us from precisely locating the searched QR template, detecting a vague blob instead.

Therefore, we finally chose CAMShift as our tracking method, as it only requires a colour marker on the top of the robot to work, being barely affected by the image resolution. The robot is located in the original image and then that coordinates are projected to the transformed image with the aid of the perspective correction matrix.

This method's main drawbacks are the lack of robustness against lighting changes (e.g. sunlight vs. laboratory's artificial light) and the loss of accuracy when the robot is far away from the camera (tracking marker is very small and subject to noise).

Another known issue is that, as we are tracking a colour marker placed on the top of the robot, it has a certain height (11 cm) that, along with the perspective correction based on landmarks, produces an evident measurement error in the localization of the robot (it is located "higher" in the Y-axis of the transformed image than where it really is). This measurement error was later reduced when, instead of using landmarks to compute the perspective correction, the reference marks were placed on top of begs of the same height as the robot (the tracking marker and the marks are therefore, placed within the same plane). A sample image of this new perspective correction is provided in figure 4.7. If we aimed to minimize this error as much as possible, it would be required to place the overhead camera right above the workspace so that locating the marker is completely equivalent to locating the robot.

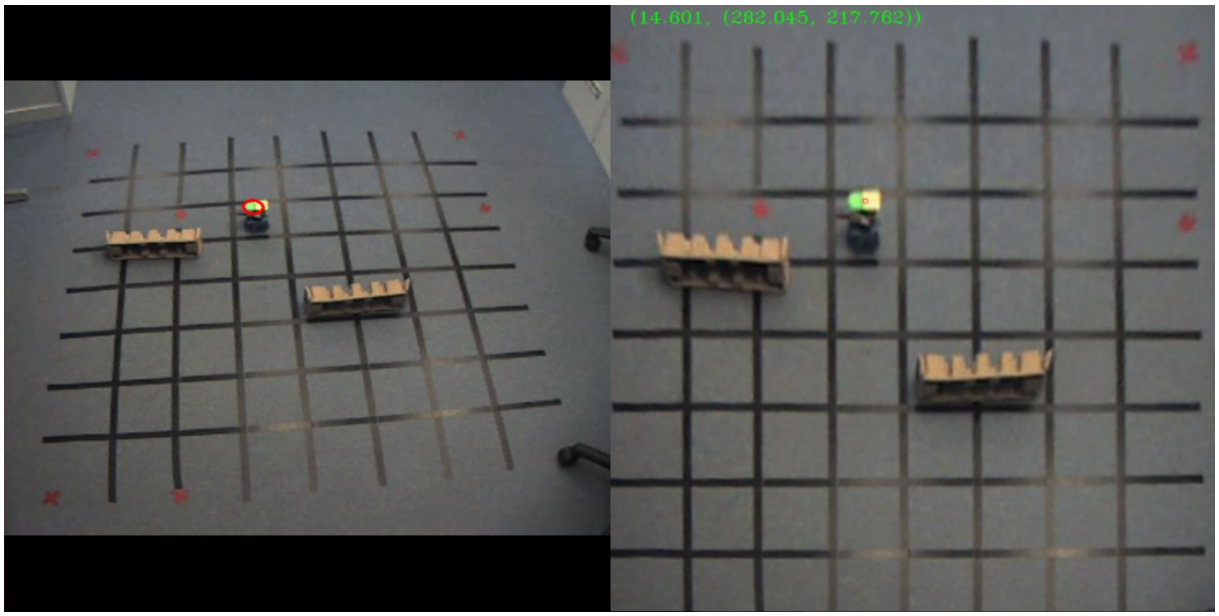


Figure 4.7: Tracking sample image. The exact location of the robot in the transformed image (red circle) is the middle point between the green and the yellow markers; this image also offers feedback to the user indicating the current angle of the robot (14.601 degrees) and its current coordinates, (282.045, 217.762) (top-left corner is the origin of coordinates). Obstacles are better managed with this new configuration, as they are now in the same plane as the robot's tracking marker.

4.3.1 Orientation

The localization problem was solved with CAMShift, but the orientation problem still remained. In [42], a two-colour marker was used to solve this issue: one of the colours was used as a tracking reference whilst the other was used as an orientation reference.

Comparing the location of one colour with respect to the other, it was possible to calculate the angular orientation of the marker and thus, of the robot.

This same idea was implemented using a two-colour marker (green: localization, yellow:orientation) and properly applying a yellow mask to the original image in order to detect and extract the yellow contour which is placed the nearest to the green contour given by CAMShift. Orientation is then computed as the angular inclination of the segment determined by the centres of green and yellow contours after correcting their perspective (thus, orientation is computed in the transformed image). A close-up of the marker is supplied in figure 4.8.

Figure 4.8: Close-up of the two-colour marker. The green colour is used for localization purposes while the yellow one is employed to compute robot's orientation.

4.4 Arduino program

The script uploaded to the Arduino board was implemented to be as simple as possible, completely relying the control of the robot on the orders received by the system via Bluetooth. The main loop of this program reads two integers from the serial port that represent values in microseconds (μs) and then writes them to the servos, controlling the shafts accordingly. On standard servos a parameter value of 1000 μs is fully counter-clockwise, 2000 μs is fully clockwise, and 1500 μs is in the middle (stop position). The robot controller is in charge of generating these values accordingly to produce the desired motion thus it takes into account the mentioned range of possible values and the fact that the servos move in opposite directions. Additionally, certain out-of-range values switch on a LED, being sent whenever the robot reaches its destination.

4.4.1 Bluetooth communication

Bluetooth technology has been evaluated in order to prove its feasibility of transmitting order motions from the robot to the system. The test consisted in timing the elapsed time between sending an order to the robot and receiving its response, repeated 50 times. The result was that the mean time elapsed was 0.00033782 seconds, with a standard deviation of $3.66069337694e-05$ seconds. Therefore, the suitability of this technology to achieve real time control has been demonstrated.

4.5 Robot control

The robot controller module is in charge of generating the suitable values for the servos that lead the robot towards its destination. Motion values are dependent on an error measurement that takes into account positional and angular errors of the robot, calculated as a difference between robot's current and desired pose. These errors have been integrated in the mathematical equations that model the control of the robot. Thus, after applying trial and error trying to keep these equations as simple as possible (few parameters), the resulting equations are:

$$\frac{dx}{dt} = p * \|\mathbf{dp} - \mathbf{cp}\| + c \tag{4.1}$$

$$\frac{d\theta}{dt} = a * (da - ca) \tag{4.2}$$

where in 4.1:

- $\frac{dx}{dt}$ models the speed of the robot. This value will always be positive.
- $\|\mathbf{dp}-\mathbf{cp}\|$ (from now on, positional error) is the Euclidean distance between desired ($\mathbf{dp}$) and current ($\mathbf{cp}$) positions, represented as 2D coordinates in the transformed image. Current position is obtained from the tracker, whilst desired position is fixed over time.
- p is a parameter associated to the positional error. From now on, referred to as positional ratio.
- c is a constant whose value produces the minimum effective movement of the robot.

and in 4.2:

- $\frac{d\theta}{dt}$ models the angular speed of the robot. This value can either be positive or negative depending on the angular error.

- $(da - ca)$ (from now on, angular error) is the angular difference between desired (**da**) and current (**ca**) angular orientations. Current angle is obtained from the tracker, whilst the aim angle is recalculated in each iteration as the angular inclination of the segment defined by current and desired positions.

- a is a parameter associated to the angular error. From now on, referred to as angular ratio.

The values obtained from these two equations are conveniently added to the middle value (1500 μs), resulting in a motion value for each servo. Both generated values are sent to the robot, producing an effective motion with the aim of leading it towards its destination. Tolerance values for angle and position were defined: if the positional error is lower than 10 pixels, the value of the first equation is zero; if the absolute value of the angle error is lower than 10 degrees, the value of the second equation is zero. In case the positional error is below positional tolerance, the robot is estimated to have reached its destination.

It is worth mentioning that control equations have been defined seeking simplicity and generality. Using higher degree equations we could have achieved a better control over robot's speed and precision, but the number of parameters would have multiplied (e.g. speed acceleration, angular speed partially depends on positional speed and vice versa, etc.), highly increasing the complexity of the problem as they would need to be properly adjusted. We preferred a simple control method with only two parameters (angular and positional ratios) to adjust, allowing a battery testing for this purpose.

5

Experimental results

When the system was completely developed, we started the testing phase applying a trial and error method, adjusting by hand the parameters (angular and positional ratio) in the equations that model the control of the robot. At this point, a rough and basic control of the robot was achieved. With the aim of improving this primitive control, we designed a battery testing that would give us a general idea about a suitable range of values for those ratios.

5.1 Battery testing

The test consisted in moving within the workspace from one fixed point A to another fixed point B and then return to A. Points A and B were separated 353.55 pixels ($\approx$ 60.5 cm) and the positional tolerance was 10 pixels ($\approx$ 1.7 cm). Here, three variable parameters were considered: angular ratio, positional ratio and update period. The update period dictates the time interval that has to pass before the robot controller module receives an update of the robot's current position and orientation. If the value of this period is high, the robot controller will send out of date motion orders to the robot, making it trace inconsistent trajectories that most of the times will end up in failure (the robot is unable to find its destination).

Described step by step, the testing was conducted as follows:

1. Starting at A, the robot spins a random angle.
2. The robot starts moving towards point B using the set of parameters (combination of three values) that is being tested. In this action, the elapsed time (difference

between arrival and departure times) is stored; if this elapsed time surpasses a certain value (50 seconds), a timeout triggers, stopping this testing and making the robot return to A.

3. The robot reaches B and spins a random angle.

4. The robot starts moving towards point A using the same set of parameters. As before, the elapsed time is stored and if it surpasses a certain value (50 seconds), a timeout triggers, stopping this testing and making the robot return to A.

5. The result of this test is the sum of the partial times required to perform both trajectories.

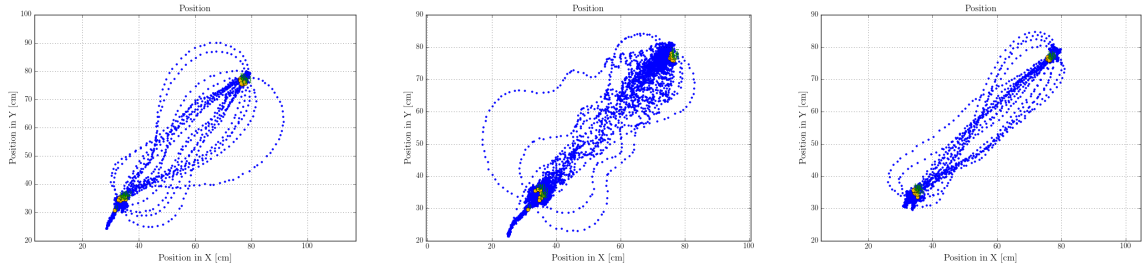
The testing is repeated 10 times, resulting in a mean time and standard deviation for each combination of values for control parameters. We defined an exhaustive battery testing that tests a total of 140 combinations of values: angular ratio, 5 values (1.5, 3.0, 4.5, 6.0, 7.5); positional ratio, 4 values (0.35, 0.45, 0.55, 0.65); update period, 7 values (0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5 seconds). Table 5.1 gathers a selection of value combinations that obtained good results in this testing. The full table of results from the battery testing is provided in Appendix B.

#Test	Update Period	Angular Ratio	Positional Ratio	Mean Time	Standard Deviation
1	0.1	3.000000	0.550000	7.553793	0.718830
2	0.1	4.500000	0.650000	7.444231	1.162515
3	0.15	3.000000	0.550000	7.296337	0.599238
4	0.15	3.000000	0.650000	7.320155	1.021303
5	**0.2**	**3.000000**	**0.550000**	**6.884031**	**0.498419**
6	0.2	3.000000	0.650000	7.245944	1.211135
7	0.25	3.000000	0.550000	7.106207	0.554916
8	0.25	3.000000	0.650000	6.541701	0.645725
9	0.3	3.000000	0.550000	7.066813	0.459982
10	0.3	3.000000	0.650000	6.749894	1.266400
11	0.4	3.000000	0.550000	7.042388	0.464557
12	0.4	3.000000	0.650000	6.904928	0.812075
13	0.5	3.000000	0.550000	6.830731	0.577281

Table 5.1: Battery testing results - Best parameter values. The criteria followed to make the selection of the best values were choosing those which had the lowest mean time and standard deviation.

From this selection, the combination of values nearest to the optimum might be the fifth: update period = 0.2, angular ratio = 3.0, and positional ratio = 0.55. We preferred this combination over other interesting ones (like number 8, 10 or 13) because it has a lower update period than them, making it more suitable for pursuing a mobile target. Figure 5.1 shows a comparison between testing trajectories traced using different parameter values.

It should be noted that a combination of parameter values that is optimum for any



(0.25, 3.0, 0.65) (0.25, 7.5, 0.65) (0.2, 3.0, 0.55)

Figure 5.1: Testing - Position log. The first picture represents wide and extremely fast trajectories, whilst the second illustrates chaotic and slow ones. The ratios of the third picture are the ones we have chosen, presenting fast and more adjusted trajectories.

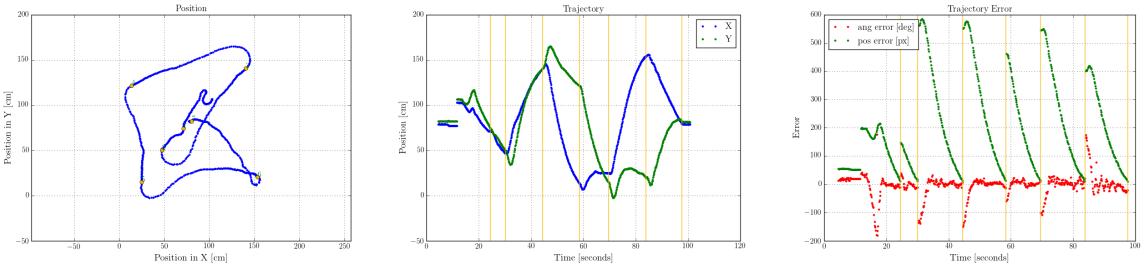
trajectory does not exist. Given a set of points describing a path, it is possible to find the values that optimize the time spent in completing it for example by running a battery test like the one we have conducted. However, these values might not be the optimum ones for another arbitrary trajectory. As we could see during the testings, for certain values, the robot could lose track and never reach the objective due to the random angle from which it started moving. Moreover, the optimality of the values changes depending on the purpose of the control: if we aim to reach a certain position or trace a particular path, we will be interested in values that minimize the error made at each step, whereas if we aim to pursue a moving target, in general we will prefer values that maximize the speed. Thus, choosing parameter values is usually a trade-off between speed (minimum elapsed time to reach your destination) and precision (minimum error made whilst reaching your destination).

5.2 Applications

As a result of the battery testing, we had a combination of parameter values that were estimated to produce good results in most of the situations. We expose in this section three different applications for our system that make use of this new control.

5.2.1 Fixed trajectory

Given a set of points, we expect the robot to move from one to another tracing the corresponding trajectory. A balanced relationship between precision and speed is desired in this application. A sample position log of this type of trajectory is provided in figure 5.2.

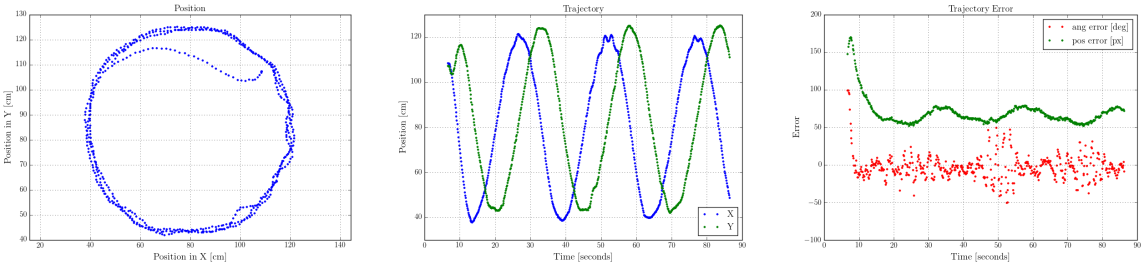


Position log | Position vs time | Error

Figure 5.2: Fixed trajectory example. The first picture is the trajectory traced by the robot whilst moving from one point to another (destination points are represented as yellow dots), the second picture represents those positions over the time and the third one shows the error made (notice how quickly the error decreases as the robot gets closer to the objectives).

5.2.2 Mobile target

In this application, the robot pursues a mobile target that is moving describing a regular trajectory (a circumference) with the aim of not losing track of it, stabilizing at a certain distance to the target and maintaining that positional error over the time. Here, we had to duplicate the positional ratio to achieve this goal, otherwise, the robot is unable to get even close to the mobile target. A sample position log of this trajectory is represented in figure 5.3.



Position log | Position vs time | Error

Figure 5.3: Mobile target trajectory example. The first picture is the trajectory traced by the robot whilst pursuing the mobile target that is tracing a circumference, the second picture represents those positions over the time (notice how regular they are, just like cosine and sine functions) and the third one shows the error made (both angular and positional errors fluctuate around an optimum stabilization value).

5.2.3 Integration with the block-programming platform

As a last application of our system, we performed the integration with the block-programming platform ARDUAMBOT [1]. In this context, precision prevails over speed, as the robot has to precisely move from one cell of the grid drawn on the workspace to another. An example of this application can be seen in figures 5.4 and 5.5.

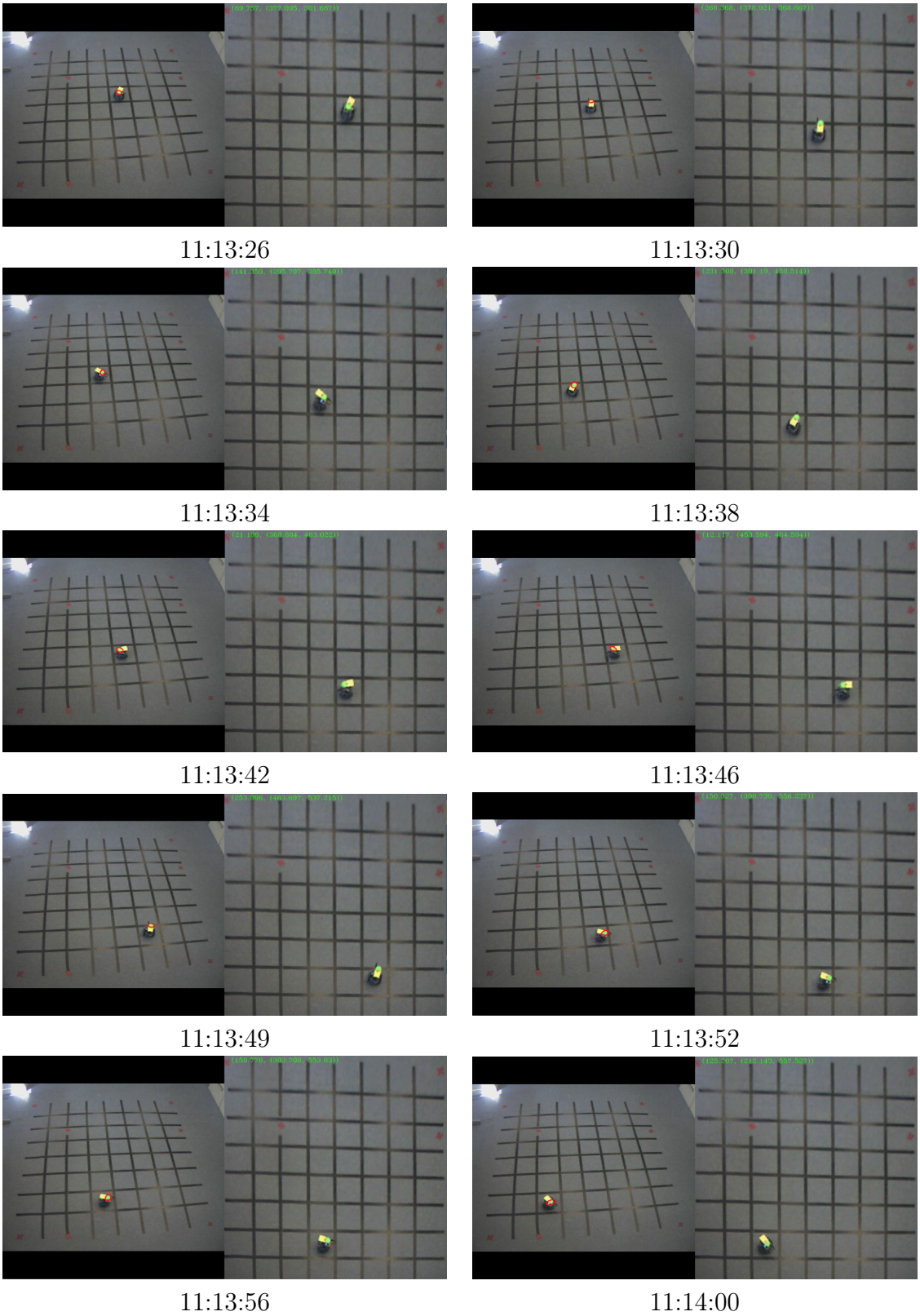


Figure 5.4: Project trajectory - Footage frames. The succession of pictures represent the evolution of the robot position over the time, as it traces the path specified in ARDUAMBOT, figure 5.5. The timestamp of each frame appears below it.

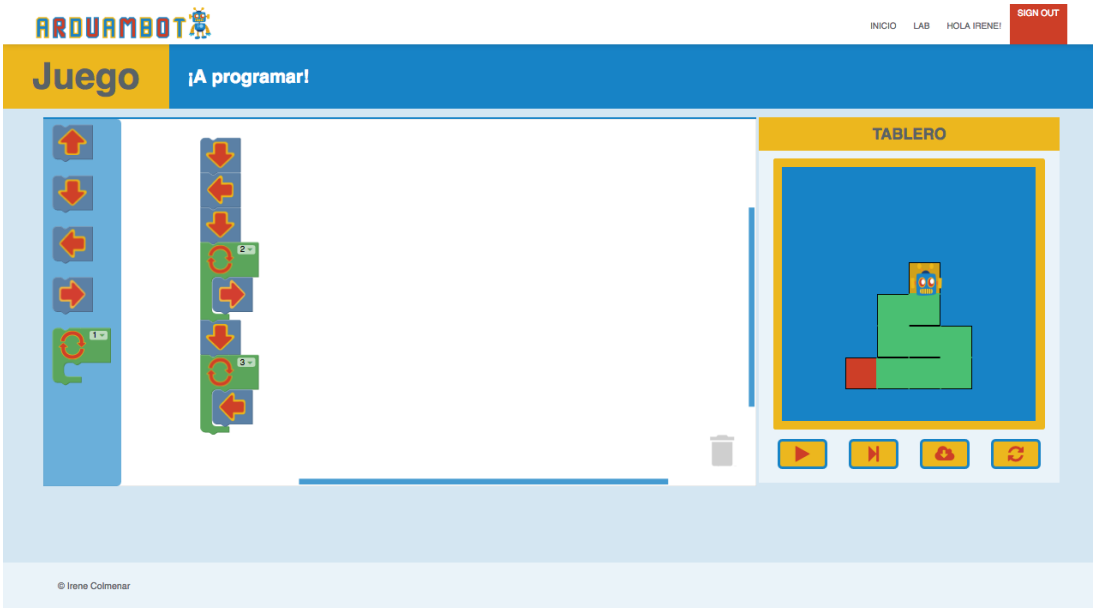


Figure 5.5: Trajectory defined from ARDUAMBOT. We defined an example trajectory from the block-programming platform in order to be followed by the robot over the grid drawn on the workspace which simulates the cells of the board in ARDUAMBOT.

5.3 Validation of the system

The validation of the system comes as a result of the exhaustive battery testing we have conducted. The experimental results obtained prove the feasibility of the whole system since it is able of controlling the robot from its current position to any desired pose making use of some suitable set of control parameter values extracted from that testing. The level of accuracy achieved with this control is quite significant, as destination points are reached with a maximum positional error of 10 pixels (1.7 cm), which demonstrates the effectiveness of the proposed system.

Three different applications for the system have been developed, each one producing satisfactory results on their corresponding tasks, which highly reinforces the validity of the proposed system.

6

Conclusions and Future Work

This project addresses the visual servoing problem of controlling the motion of a robot by means of visual feedback provided by an overhead camera. In this document we propose a solution to this problem designing and developing a software platform that can be easily set up using low cost hardware components. The system is capable of precisely tracking a robot, find out its current orientation and send it the most suitable motion commands in order to lead it to a desired position within a planar workspace. We have shown how it is possible to develop a system that, using a simple tracking algorithm (CAMShift) and simple motion control equations, obtains more than acceptable results.

The validity of the system comes from the experimental results obtained in the conducted battery testing (140 tests, each one repeated 10 times = 1400 executions), which provided values that polished the motion equations. With an improved control of the robot, we implemented and tested three different applications for our system: following a fixed trajectory, pursuing a mobile target and integrating our system with a block-programming platform, in charge of sending sets of points in order to make the robot follow them over a grid drawn on the workspace. These applications serve to clearly demonstrate its feasibility and effectiveness in real world problems and thus, validating the proposed system.

6.1 Future Work

Our future in this area will address the enhancement of the current tracking method, changing from a colour-based tracker to a template matching one (perhaps being necessary to improve our hardware to make it possible). A template matching tracker capable of detecting (for example) QR codes would be able to recognize the detected template

associated to a particular robot, identifying it in the scene.

Therefore, an immediate improvement of the system is the extension of the control to multiple robots. Controlling multiple robots allow the development of cooperative strategies that aim to solve more complex tasks. For example, instead of just pursuing a mobile target, several robots could cooperate splitting up in different directions in order to surround the mobile target and capture it.

Control equations are currently quite simple, being a pending task to substitute them by more complex ones in the future. Another method to enhance current control would be adding artificial intelligence (e.g. neural networks) so that we could train them in an automatic way, discarding the use of complex mathematical equations dependent on lots of unknown parameters.

Our ultimate goal is to be able to plan and control multiple robots, extending the effective workspace by adding several overhead cameras. If properly calibrated, the transformed images of all the cameras could be put together to form a huge workspace. This way, when a camera loses track of the target, it forwards the last available information about robot's position to the next camera, which has to make use of that information to resume the tracking.

Bibliography

[1] Irene Colmenar Guindal. "Programación por Bloques para un Robot Móvil". In: *UAM - Trabajo de Fin de Grado* (2016). URL: Notpublishedyet.

[2] Guilherme N. DeSouza and Avinash C. Kak. "Vision for Mobile Robot Navigation: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume:24 , Issue:2)* (2002), pp. 237–267. DOI: 10.1109/34.982903. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=982903.

[3] Matthew Berkemeier et al. "Visual Servoing of an Omni-Directional Mobile Robot for Alignment with Parking Lot Lines". In: *Robotics and Automation, 2002. Proceedings. ICRA '02. IEEE International Conference on (Volume:4)* (2002), pp. 4204–4210. DOI: 10.1109/ROBOT.2002.1014412. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1014412.

[4] Rahul S. Rao, Vijay Kumar, and Camillo J. Taylor. "Planning and Control of Mobile Robots in Image Space from Overhead Cameras". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (2005), pp. 2185–2190. DOI: 10.1109/ROBOT.2005.1570437. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1570437.

[5] Rahul Rao, Camillo Taylor, and Vijay Kumar. "Experiments in Robot Control from Uncalibrated Overhead Imagery". In: *ISER* (2006). URL: http://www.cis.upenn.edu/~cjtaylor/PUBLICATIONS/pdfs/RaoSpringer06.pdf.

[6] Min-Fan Ricky Lee et al. "Visual Sensor Integration on Servoing the Autonomous Mobile Robot". In: *System Integration (SII), 2013 IEEE/SICE International Symposium on, Kobe* (2013), pp. 185–189. DOI: 10.1109/SII.2013.6776713. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6776713.

[7] Florent Nageotte et al. "Visual Servoing-Based Endoscopic Path Following for Robot-Assisted Laparoscopic Surgery". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China* (2006), pp. 2364–2369. DOI: 10.1109/IROS.2006.282647. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4058740.

[8] Alexandre Krupa et al. "Autonomous 3-D Positioning of Surgical Instruments in Robotized Laparoscopic Surgery Using Visual Servoing". In: *IEEE Transactions on Robotics and Automation (Volume:19 , Issue:5)* (2003), pp. 842–853. DOI: 10.1109/TRA.2003.817086. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1236757.

[9] Andrea Ranftl et al. “Enhancing Hybrid Force/Velocity Control with Visual Servoing in Robot-Assisted Total Knee Arthroplasty”. In: *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006. BioRob 2006, Pisa* (2006), pp. 141–146. DOI: 10.1109/BIOROB.2006.1639074. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1639074.

[10] Ye Shi et al. “Modeling and Simulation of Space Robot Visual Servoing for Autonomous Target Capturing”. In: *2012 IEEE International Conference on Mechatronics and Automation, Chengdu* (2012), pp. 2275–2280. DOI: 10.1109/ICMA.2012.6285698. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4058740.

[11] Chun-Hua Chang, Shao-Chen Wang, and Chieh-Chih Wang. “Vision-Based Cooperative Simultaneous Localization and Tracking”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on, Shanghai* (2011), pp. 5191–5197. DOI: 10.1109/ICRA.2011.5980505. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5980505.

[12] Yang Li and Shu Jun. “RoboCup Small Size League: Design of Group IDs and Algorithm for Posture Parameters”. In: *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on (Volume:3), Nanchang* (2009), pp. 160–163. DOI: 10.1109/IITA.2009.59. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5369490.

[13] Seth Hutchinson, Gregory D. Hager, and Peter I. Corke. “A Tutorial on Visual Servo Control”. In: *IEEE Transactions on Robotics and Automation (Volume:12 , Issue:5)* (1996), pp. 651–670. DOI: 10.1109/70.538972. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=538972.

[14] J. Shaw and K.Y. Cheng. “Object Identification and 3-D Position Calculation Using Eye-in-Hand Single Camera for Robot Gripper”. In: *2016 IEEE International Conference on Industrial Technology (ICIT), Taipei* (2016), pp. 1622–1625. DOI: 10.1109/ICIT.2016.7475004. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7475004.

[15] Eliza beth A. Croft Simon Léonard and James J. L. “Planning Collision-Free and Occlusion-Free Paths for Industrial Manipulators with Eye-to-Hand Configuration”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, St. Louis, MO* (2009), pp. 5083–5088. DOI: 10.1109/IROS.2009.5354019. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5354019.

[16] Abdul Muis and Kouhei Ohnishi. “Eye-to-hand Approach on Eye-in-hand Configuration within Real-time Visual Servoing”. In: *IEEE/ASME Transactions on Mechatronics (Volume:10 , Issue:4)* (2005), pp. 404–410. DOI: 10.1109/TMECH.2005.852397. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1297945.

[17] Ren C. Luo et al. "Hybrid Eye-to-hand and Eye-in-hand Visual Servo System for Parallel Robot Conveyor Object Tracking and Fetching". In: *IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society, Dallas, TX* (2014), pp. 2558–2563. DOI: 10.1109/IECON.2014.7048866. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7048866.

[18] Wen-Chung Chang and Chia-Kai Shao. "Hybrid Eye-to-Hand and Eye-in-Hand Visual Servoing for Autonomous Robotic Manipulation". In: *SICE Annual Conference 2010, Proceedings of, Taipei* (2010), pp. 415–422. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5602381.

[19] François Chaumette and Seth Hutchinson. "Visual Servo Control Part I: Basic Approaches". In: *IEEE Robotics & Automation Magazine (Volume:13 , Issue:4)* (2006), pp. 82–90. DOI: 10.1109/MRA.2006.250573. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4015997.

[20] Chia-Hsien Louis Chen and Min-Fan Ricky Lee. "Global Path Planning in Mobile Robot using Omnidirectional Camera". In: *Consumer Electronics, Communications and Networks (CECNet), 2011 International Conference on, XianNing* (2011), pp. 4986–4989. DOI: 10.1109/CECNET.2011.5768666. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5768666&tag=1.

[21] Roel Pieters. "Visual Servo Control". In: *Eindhoven University of Technology, The Netherlands* (2012). URL: http://www.ics.ele.tue.nl/~heco/courses/EmbeddedVisualControl/visual-servo-control.pdf.

[22] Emanuele Trucco and Konstantinos Plakas. "Video Tracking: A Concise Survey". In: *IEEE Journal of Oceanic Engineering (Volume:31 , Issue:2)* (2006), pp. 520–529. DOI: 10.1109/JOE.2004.839933. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1708000.

[23] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *J. Basic Eng 82(1)* (1960), pp. 35–45. DOI: 10.1115/1.3662552. URL: http://fluidsengineering.asmedigitalcollection.asme.org/article.aspx?articleid=1430402.

[24] Jing Ren and Jie Hao. "Mean Shift Tracking Algorithm Combined with Kalman Filter". In: *Image and Signal Processing (CISP), 2012 5th International Congress on, Chongqing* (2012), pp. 727–730. DOI: 10.1109/CISP.2012.6469936. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6469936.

[25] Xin Li et al. "A Multiple Object Tracking Method Using Kalman Filter". In: *Information and Automation (ICIA), 2010 IEEE International Conference on, Harbin* (2010), pp. 1862–1866. DOI: 10.1109/ICINFA.2010.5512258. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5512258.

[26] Daniel Scharstein, Richard Szeliski, and Ramin Zabih. "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms". In: *Stereo and Multi-Baseline Vision, 2001. (SMBV 2001). Proceedings. IEEE Workshop on, Kauai, HI* (2001), pp. 131–140. DOI: 10.1109/SMBV.2001.988771. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=988771.

[27] Mahbub Murshed, M. Ali Akber Dewan, and Oksam Chae. "Moving Object Tracking - A Parametric Edge Tracking Approach". In: *Computers and Information Technology, 2009. ICCIT '09. 12th International Conference on, Dhaka* (2009), pp. 471–476. DOI: 10.1109/ICCIT.2009.5407285. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5407285.

[28] Takashi Anezaki et al. "Development of a Human-Tracking Robot Using QR Code Recognition". In: *Frontiers of Computer Vision (FCV), 2011 17th Korea-Japan Joint Workshop on, Ulsan* (2011), pp. 1–6. DOI: 10.1109/FCV.2011.5739699. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5739699.

[29] Zhiquan Feng et al. "Behavioral Model Tracking of Hand Gestures". In: *2015 International Conference on Virtual Reality and Visualization (ICVRV), Xiamen* (2015), pp. 101–108. DOI: 10.1109/ICVRV.2015.11. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7467220.

[30] Hao Jiang and Mark S. Drew. "A Predictive Contour Inertia Snake Model For General Video Tracking". In: *Image Processing. 2002. Proceedings. 2002 International Conference on (Volume:3)* (2002), pp. 413–416. DOI: 10.1109/ICIP.2002.1038993. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1038993.

[31] Raed Almomani, Ming Dong, and Zhou Liu. "Learning Good Features To Track". In: *Machine Learning and Applications (ICMLA), 2014 13th International Conference on, Detroit, MI* (2014), pp. 373–378. DOI: 10.1109/ICMLA.2014.66. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7033143.

[32] Ido Leichter, Michael Lindenbaum, and Ehud Rivlin. "A Probabilistic Framework for Combining Tracking Algorithms". In: *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on (Volume:2)* (2004), pp. 445–451. DOI: 10.1109/CVPR.2004.1315198. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1315198.

[33] An Guocheng, Li Hongyan, and Li ming. "Novel Multi-Window Tracking Based on Model Update". In: *Computational Intelligence and Design (ISCID), 2014 Seventh International Symposium on (Volume:1), Hangzhou* (2014), pp. 3–6. DOI: 10.1109/ISCID.2014.41. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7064065&tag=1.

[34] Dorin Comaniciu and Peter Meer. "Robust Analysis of Feature Spaces: Color Image Segmentation". In: *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on, San Juan* (1997), pp. 750–755. DOI: 10.1109/CVPR.1997.609410. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=609410.

[35] Dorin Comaniciu and Peter Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume:24 , Issue:5)* (2002), pp. 603–619. DOI: 10.1109/34.1000236. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1000236.

[36] Xingmei Wang et al. "Mean-Shift Tracking Algorithm based on Kalman Filter using Adaptive Window and Sub-blocking". In: *Intelligent Control and Automation (WCICA), 2014 11th World Congress on, Shenyang* (2014), pp. 5438–5443. DOI: 10.1109/WCICA.2014.7053643. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7053643.

[37] Yan Yang et al. "An Improved Mean Shift Object Tracking Algorithm Based on ORB Feature Matching". In: *The 27th Chinese Control and Decision Conference (2015 CCDC), Qingdao* (2015), pp. 4996–4999. DOI: 10.1109/CCDC.2015.7162819. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7162819.

[38] Ethan Rublee et al. "ORB: an efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision, Barcelona* (2011), pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6126544.

[39] Gary R. Bradski. "Computer Vision Face Tracking For Use in a Perceptual User Interface". In: (1998). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.7673.

[40] Rustam Stolkin et al. "Efficient visual servoing with the ABCshift tracking algorithm". In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on, Pasadena, CA* (2008), pp. 3219–3224. DOI: 10.1109/ROBOT.2008.4543701. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4543701.

[41] Wael Mohamed Yousf, Osama Mohamed Elmowafy, and Ibrahim Ali Abdl-Dayem. "C18. Modified CAMShift Algorithm for Adaptive Window Tracking". In: *Radio Science Conference (NRSC), 2012 29th National, Cairo* (2012), pp. 301–308. DOI: 10.1109/NRSC.2012.6208536. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6208536.

[42] Carlos García-Saura. "Cooperative Strategies for the Detection and Localization of Odorants with Robots and Artificial Noses". In: *UAM - Bachelor's Thesis* (2014). URL: https://repositorio.uam.es/xmlui/bitstream/handle/10486/662644/Garc%C3%ADa_Saura_Carlos_tfg.pdf?sequence=1.

Appendices

A

Hardware Specifications

A.1 SpringRC SM-S4303R Servo Datasheet

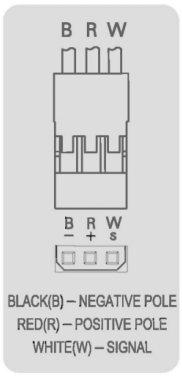
SPRINGRC®
www.springrc.com

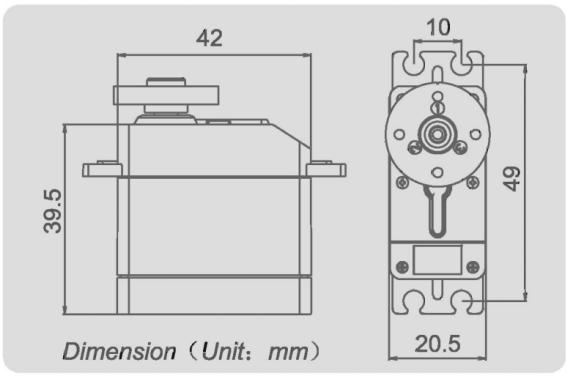
43R Servo(360° Rotation) Specification

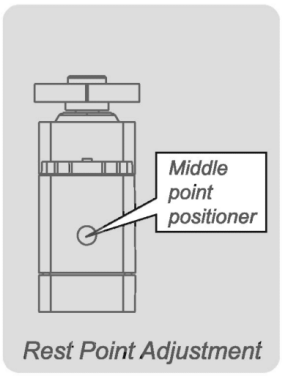
Thank you for choosing Spring Model's product

MODEL	TYPE	WEIGHT		4.8V			6V			DESCRIPTION	
		g	oz	SPEED	TORQUE		SPEED	TORQUE		GEAR	BEARING
				r/min	kg.cm	oz.in	r/min	kg.cm	oz.in		
SM-S4303R	Analog	44	1.55	60	3.3	45.8	70	4.8	66.7	1Metal Gear+ 4Plastic Gear	2
SM-S4306R		44	1.55	60	5.0	69.4	50	6.2	86.1	1Metal Gear+ 4Plastic Gear	2
SM-S4309R		60	2.12	58	7.9	109.7	49	8.7	120.8	Metal Gear	2
SM-S4315R		60	2.12	62	14.5	201.4	53	15.4	213.9	Metal Gear	2

▲ 43R Robot series servo controled via analog signal(PWM),stopped via middle point positiner.
▲ Standard interface(like JR)with 30cm wire.
▲ Rotation and Rest Point Adjustment:when analog signal inputs,servo chooses orientation according to impulse width.when intermediatevalue of impluse width is above 1.5ms, servo is clockwise rotation,conversely,anticlockwise.Rest point need use slotted screwdriver to adjust the positioner carefully.Servo stopped rotation when the input signal is equivalent to impluse width.
▲ Please choose correct model for your application.
Caution: Torque over-loaded will damage the servo's mechanism.
▲ Keep the servo clean and away from dust, corrosive gas and humid air.
▲ Without further notification when some parameters slightly amend for improving quality.







SPRING MODEL ELECTRONICS Co.,LTD. MADE IN CHINA XBBS4306R119 V1.0

RoHS CE

Figure A.1: SpringRC SM-S4303R - Datasheet.

A.2 BQ ZUM BT-328 Schematics

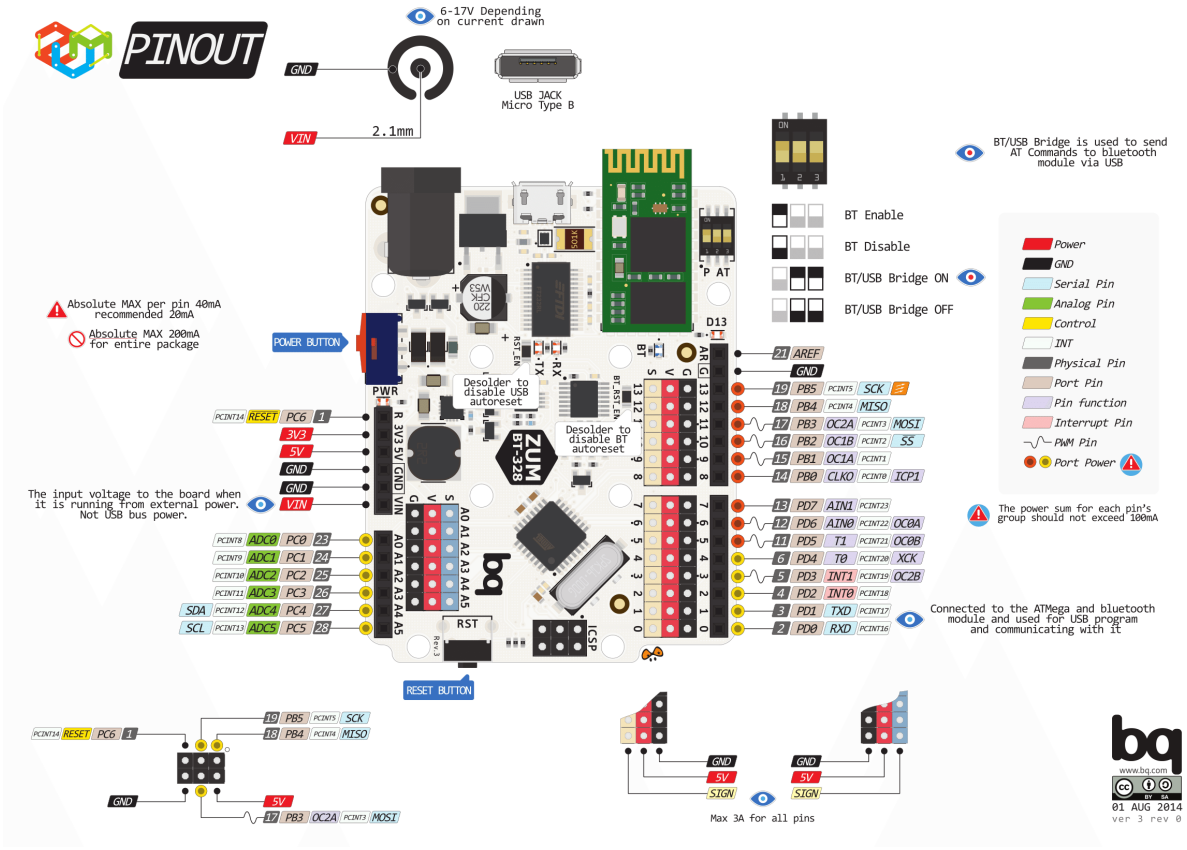
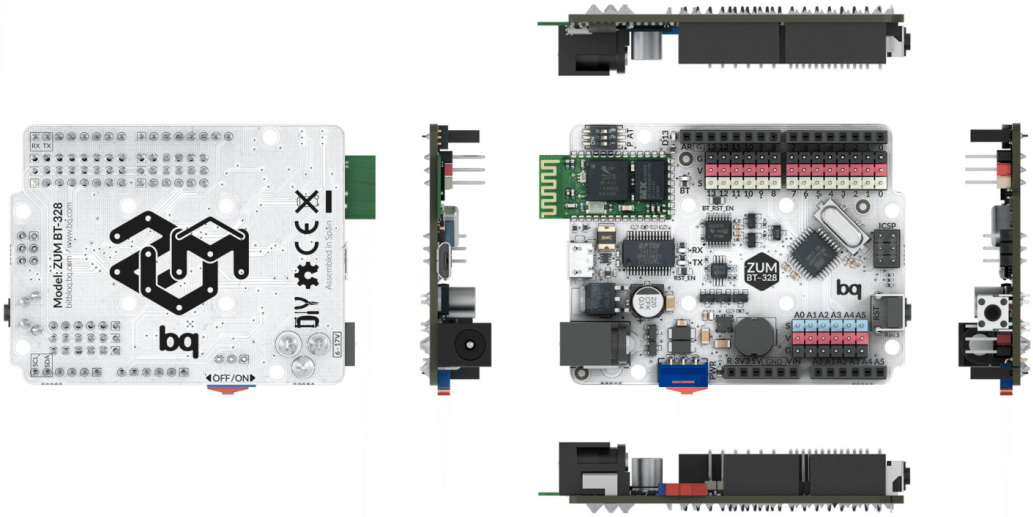


Figure A.2: BQ ZUM BT-328 - Schematics.

bq ZUM BT-328

bq

bq ZUM BT-328



CARACTERÍSTICAS TÉCNICAS

EAN: 8436545518649 - P/N: H000163

Descripción

bq ZUM permite crear circuitos electrónicos capaces de recibir información del entorno a través de sensores y realizar acciones con actuadores y motores.

Características

Módulo bluetooth integrado
Hasta 3 amperios de salida
Interruptor y botón de reseteo
100% compatible con Arduino

Dimensiones y peso

Dimensiones: 75 x 55 x 15 mm
Peso: 23 g

Especificaciones

Microcontrolador: ATmega328P
Voltaje de entrada: 6 - 17V
Corriente de salida: 3.3V (50mA) - 5V (3.2A)
Entradas analógicas: 6
E/S Digitales: 14
CPU: 16 MIPS
Interfaces: Bluetooth 2.1, USB, ICSP, TTL UART, SPI e I2C.
Baudios Bluetooth: 19200 bps
Señales LED: 5 - RX, TX, Power, D13 y Bluetooth.

8 436545 518649

bq ZUM BT-328

Fabricado en España

Figure A.3: BQ ZUM BT-328 - Technical Specifications.

A.3 D-Link DCS-3110 Camera Specifications

Figure A.4: D-Link DCS-3110 - Dimensions.

D-Link®

LIVE VIDEO MONITORING FROM ANYWHERE

WHAT THIS PRODUCT DOES

The DCS-3110 Megapixel PoE Network Camera includes a 1.3 mega-pixel progressive scan CMOS sensor capable of capturing high-resolution images in a larger coverage area, making it significantly better than an analog CCTV camera. The DCS-3110 can effectively reduce motion blur and distorted, jagged edges in recorded images, and is usable for both day and night applications. In addition, it supports MPEG-4 multicast streaming - a bandwidth efficient way of providing streaming content to multiple users through an IP address, as well as allowing for dual-streaming for live monitoring and recording simultaneously. The DCS-3110 comes with a built-in 802.3af compliant PoE module for easy setup and flexible placement in an indoor or outdoor surveillance site.

REMOTE MONITORING OPTION

The DCS-3110 features 3G mobile video support, enabling you to view a live video feed from your camera on a compatible 3G mobile phone or PDA anywhere in your 3G coverage area. 3G mobile video support allows you to monitor your camera from a remote location without a computer. D-Link's D-ViewCam software is also included to let you manage up to 32 cameras simultaneously from your computer, automatically receive e-mail alerts, and record video to your hard drive and network storage when motion is detected.

TECHNICAL SPECIFICATIONS

NETWORK PROTOCOL SUPPORT
- IPv4, TCP/IP, RTSP/ RTP/ RTCP, HTTP, SMTP, FTP, NTP, DNS, DHCP, UPnP, DDNS, PPPoE, IGMP, Samba client, IP Filtering, 3GPP

NETWORK INTERFACE
- IEEE 802.3/802.3u 10/100BASE-TX Ethernet port
- Supports half/full-duplex operations
- Supports 802.3x Flow Control in full-duplex mode
- Supports IEEE 802.3af PoE

VIDEO ALGORITHM SUPPORT
- JPEG for still images
- Compression: MJPEG & MPEG-4
- Streaming: Simultaneous dual-streaming
- MPEG-4 streaming over UDP, TCP, or HTTP
- MPEG-4 multicast streaming
- MJPEG streaming over HTTP

VIDEO RESOLUTION
- MJPEG video with resolution up to 1280x1024 (1.3 Megapixel)
- MPEG-4 video with resolution up to 640x480 (VGA)
- Up to 30fps at 176 x 144
- Up to 30fps at 320 x 240
- Up to 30fps at 640 x 480
- Up to 8fps at 1280x1024

VIDEO FEATURES
- Adjustable image size, quality, and bit rate
- Time stamp and text overlays
- 3 configurable motion detection windows
- Flip & mirror
- Configurable brightness
- AGC, AWB, AES

VIDEO BIT RATE
- 20K to 4M

SENSOR & LENS SPECIFICATIONS
- 1/4" 1.3 Megapixel progressive color CMOS sensor
- 4.5-10 mm megapixel lens, F1.6
- Removable IR-cut filter: Auto/Schedule/Manual
- Minimum illumination: 1.5 Lux
- Field of View
  - Diagonal: 46.5° - 102°
  - Horizontal: 37° - 80.5°
  - Vertical: 28° - 60°

EVENT MANAGEMENT
- Motion detection with configurable weekly schedule
- Event notification and upload snapshots/video clips via HTTP, SMTP, or FTP
- Multiple HTTP, SMTP, or FTP server setups
- Multiple event notification setups for flexible applications
- Multiple recording methods for easier backup

SECURITY
- Administrator and user group protected
- Password authentication
- HTTP and RTSP digest encryption

SURVEILLANCE SOFTWARE FUNCTIONS
- Remote management/control of up to 32 cameras
- Viewing of up to 32 cameras on one screen
- Supports all management functions provided in web interface
- Scheduled, motion triggered, or manual recording options

REMOTE MANAGEMENT
- Configuration accessible via web browser
- Take snapshots/video clips and save to CF card, local hard drive, or NAS via web browser
- Camera live viewing for up to 10 clients*

SYSTEM REQUIREMENTS
- Operating System: Microsoft Windows 2000, XP, or Vista
- Browser: Internet Explorer, Firefox, Netscape, Mozilla, or Opera

3GPP MOBILE SURVEILLANCE
- Handsets with 3GPP player with following software support:
  - Packet Video Player 3.0
  - QuickTime 6.5
  - Real Player 10.5

AUDIO
- Compression and bit rates:
  - GSM-AMR speech compression, bit rate: 4.75 kbps -12.2 kbps
  - MPEG-4 AAC audio encoding, bit rate: 16 kbps -128 kbps
- Interfaces: built-in microphone, external microphone input, external speaker output
- Supports two-way audio by SIP protocol
- Supports hardware and software audio mute

EXTERNAL DEVICE INTERFACE
- One D/I and one D/O for an external sensor and alarm
- RS485 interface for an external scanner
- DC12V output for an external accessory

DIAGNOSTIC LED
- 2 color LED

FRONT-END STORAGE
- CF card slot for local recording of MP4 files

POWER INPUT
- 100 - 240VAC, 50/60Hz, 12VDC, 1.25A

POWER CONSUMPTION
- Max 4W

DIMENSIONS (W x D x H)
- 80 x 166.8 x 62 mm

SURVEILLANCE

MEGAPIXEL PoE NETWORK CAMERA
DCS-3110

Figure A.5: D-Link DCS-3110 - Technical Specifications.

B

Complete battery testing results

#Test	Update Period	Angular Ratio	Positional Ratio	Mean Time	Standard Deviation
1	0.1	1.500000	0.350000	10.040239	0.343739
2	0.1	1.500000	0.450000	9.224427	0.761524
3	0.1	1.500000	0.550000	14.482262	28.754521
4	0.1	1.500000	0.650000	18.962410	26.564401
5	0.1	3.000000	0.350000	9.998775	0.280652
6	0.1	3.000000	0.450000	8.753040	0.261840
7	0.1	3.000000	0.550000	7.553793	0.718830
8	0.1	3.000000	0.650000	7.454697	2.191506
9	0.1	4.500000	0.350000	9.938567	0.379894
10	0.1	4.500000	0.450000	8.754928	0.377224
11	0.1	4.500000	0.550000	7.841523	0.710001
12	0.1	4.500000	0.650000	7.444231	1.162515
13	0.1	6.000000	0.350000	10.354206	0.452264
14	0.1	6.000000	0.450000	9.813528	1.305313
15	0.1	6.000000	0.550000	11.012477	4.375774
16	0.1	6.000000	0.650000	9.227648	3.245096
17	0.1	7.500000	0.350000	18.105664	9.491269
18	0.1	7.500000	0.450000	11.650075	2.986443
19	0.1	7.500000	0.550000	14.317783	6.980980
20	0.1	7.500000	0.650000	13.680167	5.281374
21	0.15	1.500000	0.350000	9.857696	2.327354
22	0.15	1.500000	0.450000	10.113936	2.820742
23	0.15	1.500000	0.550000	16.123187	17.196601
24	0.15	1.500000	0.650000	16.612992	16.956045

#Test	Update Period	Angular Ratio	Positional Ratio	Mean Time	Standard Deviation
25	0.15	3.000000	0.350000	9.304171	2.155256
26	0.15	3.000000	0.450000	8.629724	0.345817
27	0.15	3.000000	0.550000	7.296337	0.599238
28	0.15	3.000000	0.650000	7.320155	1.021303
29	0.15	4.500000	0.350000	9.856083	0.399743
30	0.15	4.500000	0.450000	8.707867	0.320792
31	0.15	4.500000	0.550000	7.639119	0.553557
32	0.15	4.500000	0.650000	8.930894	5.827317
33	0.15	6.000000	0.350000	10.872683	1.374928
34	0.15	6.000000	0.450000	9.565709	1.281476
35	0.15	6.000000	0.550000	10.280780	3.240804
36	0.15	6.000000	0.650000	8.625943	1.907696
37	0.15	7.500000	0.350000	16.950045	7.006865
38	0.15	7.500000	0.450000	14.643817	6.705020
39	0.15	7.500000	0.550000	14.838444	5.896846
40	0.15	7.500000	0.650000	12.210948	3.806706
41	0.2	1.500000	0.350000	9.834913	0.753496
42	0.2	1.500000	0.450000	8.917970	1.028826
43	0.2	1.500000	0.550000	16.104652	17.166895
44	0.2	1.500000	0.650000	22.023210	19.978016
45	0.2	3.000000	0.350000	9.670797	0.477236
46	0.2	3.000000	0.450000	8.463280	0.469278
47	0.2	3.000000	0.550000	6.884031	0.498419
48	0.2	3.000000	0.650000	7.245944	1.211135
49	0.2	4.500000	0.350000	9.776537	0.806997
50	0.2	4.500000	0.450000	8.635044	0.479503
51	0.2	4.500000	0.550000	7.650474	0.810440
52	0.2	4.500000	0.650000	8.985853	7.507744
53	0.2	6.000000	0.350000	11.648075	2.380287
54	0.2	6.000000	0.450000	10.290964	2.047228
55	0.2	6.000000	0.550000	11.983190	4.742083
56	0.2	6.000000	0.650000	10.163903	3.694192
57	0.2	7.500000	0.350000	18.764902	9.059060
58	0.2	7.500000	0.450000	16.553492	8.084367
59	0.2	7.500000	0.550000	19.997584	5.368085
60	0.2	7.500000	0.650000	19.397001	5.249778
61	0.25	1.500000	0.350000	10.125445	0.569527
62	0.25	1.500000	0.450000	8.899244	0.466584
63	0.25	1.500000	0.550000	13.672473	15.277020
64	0.25	1.500000	0.650000	17.865709	18.577407
65	0.25	3.000000	0.350000	9.390968	2.037780
66	0.25	3.000000	0.450000	8.630295	0.249413
67	0.25	3.000000	0.550000	7.106207	0.554916

#Test	Update Period	Angular Ratio	Positional Ratio	Mean Time	Standard Deviation
68	0.25	3.000000	0.650000	6.541701	0.645725
69	0.25	4.500000	0.350000	10.060586	0.483675
70	0.25	4.500000	0.450000	8.991272	0.489752
71	0.25	4.500000	0.550000	8.335102	1.797342
72	0.25	4.500000	0.650000	8.514466	4.766604
73	0.25	6.000000	0.350000	16.920787	7.356057
74	0.25	6.000000	0.450000	15.630881	7.578367
75	0.25	6.000000	0.550000	14.539061	6.486005
76	0.25	6.000000	0.650000	12.881549	5.637630
77	0.25	7.500000	0.350000	46.534368	9.352419
78	0.25	7.500000	0.450000	36.940035	17.018670
79	0.25	7.500000	0.550000	26.673301	5.869722
80	0.25	7.500000	0.650000	21.082555	9.282303
81	0.3	1.500000	0.350000	10.371865	0.476633
82	0.3	1.500000	0.450000	9.161075	0.637581
83	0.3	1.500000	0.550000	17.024121	18.289321
84	0.3	1.500000	0.650000	21.247236	20.347284
85	0.3	3.000000	0.350000	9.898669	0.301472
86	0.3	3.000000	0.450000	8.829619	0.270693
87	0.3	3.000000	0.550000	7.066813	0.459982
88	0.3	3.000000	0.650000	6.749894	1.266400
89	0.3	4.500000	0.350000	10.092310	0.547194
90	0.3	4.500000	0.450000	9.100304	0.605766
91	0.3	4.500000	0.550000	10.715426	4.206535
92	0.3	4.500000	0.650000	10.721634	6.887128
93	0.3	6.000000	0.350000	26.163729	13.952917
94	0.3	6.000000	0.450000	20.605601	11.891452
95	0.3	6.000000	0.550000	18.869763	7.959517
96	0.3	6.000000	0.650000	17.437023	8.717261
97	0.3	7.500000	0.350000	47.228595	8.037977
98	0.3	7.500000	0.450000	46.980002	8.829626
99	0.3	7.500000	0.550000	36.254772	9.735110
100	0.3	7.500000	0.650000	35.118755	10.150684
101	0.4	1.500000	0.350000	10.113921	0.439052
102	0.4	1.500000	0.450000	9.146708	0.551199
103	0.4	1.500000	0.550000	20.578931	19.529641
104	0.4	1.500000	0.650000	14.163716	13.793107
105	0.4	3.000000	0.350000	9.721565	1.081184
106	0.4	3.000000	0.450000	8.729771	0.338260
107	0.4	3.000000	0.550000	7.042388	0.464557
108	0.4	3.000000	0.650000	6.904928	0.812075
109	0.4	4.500000	0.350000	10.435143	0.760537
110	0.4	4.500000	0.450000	9.594657	0.990092

#Test	Update Period	Angular Ratio	Positional Ratio	Mean Time	Standard Deviation
111	0.4	4.500000	0.550000	7.975124	1.022638
112	0.4	4.500000	0.650000	8.426701	4.523640
113	0.4	6.000000	0.350000	24.768031	14.145070
114	0.4	6.000000	0.450000	19.238148	9.559501
115	0.4	6.000000	0.550000	12.194878	4.541678
116	0.4	6.000000	0.650000	12.252187	9.505079
117	0.4	7.500000	0.350000	37.483806	15.807419
118	0.4	7.500000	0.450000	29.843200	15.372727
119	0.4	7.500000	0.550000	16.217925	5.578108
120	0.4	7.500000	0.650000	12.745201	5.671606
121	0.5	1.500000	0.350000	10.179682	0.360342
122	0.5	1.500000	0.450000	8.940512	0.517842
123	0.5	1.500000	0.550000	12.221575	13.164924
124	0.5	1.500000	0.650000	18.504536	19.060555
125	0.5	3.000000	0.350000	9.697109	2.298715
126	0.5	3.000000	0.450000	8.938090	0.388662
127	0.5	3.000000	0.550000	6.830731	0.577281
128	0.5	3.000000	0.650000	7.483983	2.518732
129	0.5	4.500000	0.350000	10.966524	1.107499
130	0.5	4.500000	0.450000	10.087531	1.480457
131	0.5	4.500000	0.550000	7.662236	1.140618
132	0.5	4.500000	0.650000	7.706230	2.201788
133	0.5	6.000000	0.350000	48.988729	3.084738
134	0.5	6.000000	0.450000	56.239493	21.904461
135	0.5	6.000000	0.550000	12.816604	5.248226
136	0.5	6.000000	0.650000	10.735912	4.756757
137	0.5	7.500000	0.350000	49.884385	0.018047
138	0.5	7.500000	0.450000	52.740090	11.584118
139	0.5	7.500000	0.550000	17.451927	7.439680
140	0.5	7.500000	0.650000	13.600895	4.903930

Table B.1: Battery testing results - Full table.