POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

DEVELOPMENT OF A LOW-COST ROBOT FOR ROBOGAMES

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Coordinator: Prof. Andrea Bonarini
Collaborator: Martino Migliavacca

MS Thesis:
ANIL KOYUNCU, matricola 737109

Academic Year 2010-2011

POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

DEVELOPMENT OF A LOW-COST ROBOT FOR ROBOGAMES

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Thesis By: .....

(Anil Koyuncu)

Advisor: .....

(Prof. Andrea Bonarini)

Academic Year 2010-2011

To my family...

Sommario

L'obiettivo del progetto è lo sviluppo di un robot adatto ad implementare giochi robotici altamente interattivi in un ambiente domestico. Il robot verrà utilizzato per sviluppare ulteriori giochi robotici nel laboratorio AIR-Lab. Il progetto è stato anche utilizzato come esperimento per la linea di ricerca relativa alla robotica a basso costo, in cui i requisiti dell'applicazione e il costo costituiscono le specifiche principali per il progetto del robot. È stato sviluppato l'intero sistema, dal progetto del robot alla realizzazione del telaio, delle componenti meccaniche e elettroniche utilizzate per il controllo del robot e l'acquisizione dei dati forniti dai sensori, ed è stato implementato un semplice gioco per mostrare tutte le funzionalità disponibili. I componenti utilizzati sono stati scelti in modo da costruire il robot con il minor costo possibile. Sono infine state introdotte alcune ottimizzazini ed è stata effettuata una accurata messa a punto per risolvere i problemi di imprecisioni nati dall'utilizzo di componenti a basso costo.

Summary

Aim of this project is the development of a robot suitable to implement highly interactive robogames in a home environment. This will be used in the Robogames research line at AIRLAB to implement even more interesting games. This is also an experiment in the new development line of research, where user needs and costs are considered as a primary source for specification to guide robot development. We have implemented a full system, from building a model of the robot and the design of the chassis, mechanical components and electronics needed for implementation robot control and the acquisition of sensory data up to the design of a simple game showing all the available functionalities. The selection of the components made in a manner that will make it with the lowest cost possible. Some optimizations and tuning have been introduced, to solve the inaccuracy problem arisen, due to the adaption of low-cost components.

Thanks

I want to thank to Prof. Bonarini for his guidance and support from the initial to the final level of my thesis and also thank to Martino, Davide, Luigi, Simone and other member of AIRLAB for their support and help to resolve my problems.

Thanks to all of my friends who supported and helped my during the university studies and for thesis. Arif, Burak, Ugur, Semra, Guzide and Harun and to my house-mate Giulia and the others I forgot to mention...

Last but not least thanks to my beloved family for their endless love and continuous support.

Anneme babama ve ablama .....

Contents

List of Figures

Chapter 1

Introduction

1.1 Goals

The aim of this thesis is to develop an autonomous robot implementing the main functionalities needed for low-cost, but interesting robogames. There are some predefined constraints for the robot. One of the most important property is being the lowest cost possible. We limit the maximum cost to 250 euro. In order to satisfy this constraint, we focused on producing and reusing some of the components or choosing the components that barely work. Another constraint is the target environment of the robot, which is home environment. The size and the weight of the robot have been chosen in a way that it can move easily in home environments. The choice of the kinematics and the wheels are made according to these needs.

1.2 Context, Motivations

Finding solutions for building robots capable of moving in home environment and to cooperate with people is a subject of much study prevalent in recent years; many companies are investing in this area with the conviction that, in the near future, robotics will represent an interesting markets.

The aim of this work has been to design and implement a home-based mobile robot able to move at home and interacting with users involved in games. The presented work concerns the entire development process, from building a model of the system and the design of the chassis, mechanical components and electronics needed for implementation robot control and the acquisition of sensory data, up to the development of behaviors.

The preliminary phase of the project has been to study the kinematics problem, which led to the creation of a model system to analyze the relationship between applied forces and motion of the robot.

The work proceeded with the design of mechanical parts, using solutions to meet the needs of system modularity and using standard components as much as possible. After modeling the chassis of the robot, and having selected the wheels, the actuators have been chosen.

The hardware design has affected the choice of sensors used for estimating the state of the system, and the design of a microcontroller-based control logic.

The last part of the work consisted in carrying out experiments to estimate the position of the robot using the data from the sensors and to improve the performance of the control algorithm. The various contributors affecting the performance of the robot behavior have been tested, allowing to observe differences in performance, and alternative solutions have been implemented to cope with limitations due to low cost of HW and low computational power.

1.3 Achievements

We have been able to develop a robot that is able to follow successfully a predefined colored object, thus implementing many interesting capabilities useful for robogames. We have faced some limitations due to the low-cost constraints. The main challenges have been caused by the absence of motor encoders and low-cost optics. We have done some optimizations and tunings to overcome these limitations.

1.4 Thesis Structure

The rest of the thesis is structured as follows: In chapter 2 we present the state of the art, which is concentrated on similar applications, the techniques used, and what has been done previously. Chapter 3 is about the mechanical construction and hardware architectures. Chapter 4 is the de-

tailed description of the control, what is necessary to replicate the same work. Chapter 5 is the vision system description in detail, what has been done, which approaches are used, and the implementation. Chapter 6 concerns game design for the tests made and the evaluation of the game results. Chapter 7 concludes the presentation.

Chapter 2

State of the Art

Advances in computer engineering artificial intelligence, and high–tech evolutions from electronics and mechanics have led to breakthroughs in robotic technology [23]. Today, autonomous mobile robots can track a person's location, provide contextually appropriate information, and act in response to spoken commands.

Robotics has been involved in human lives from industry domain to daily life applications such as home helper or, recently, entertainment robots. The latter introduced a new aspect of robotics, entertainment, which is intended to make humans enjoy their lives from a various kind of view-points quite different from industrial applications [17].

Interaction with robot is thought of a relatively new field, but the idea of building lifelike machines that entertain people has fascinated us for hundreds of years since the first ancient mechanical automaton. Up to our days, there have been major improvements in the development of robots.

We will review the literature for the robots that are related with our design. We divided the review in subsections like Locomotion, Motion Models, Navigation, and Interaction and Games.

2.1 Locomotion

There exists a great variety of possible ways to move a robot, which makes the selection of a robot's approach to motion an important aspect of mobile robot design. The most important of these are wheels, tracks and legs [33].

The wheel has been by far the most popular motion mechanism in mobile robotics. It can achieve very good efficiency, with a relatively simple mechanical implementation and construction easiness. On the other hand, legs and tracks require complex mechanics, more power, and heavier hardware for the same payload. It is suitable to choose wheels for robot that is designed to work in home environment, where it has to move mainly on a plain surface.

There are three major wheel classes. They differ widely in their kinematics, and therefore the choice of wheel type has a large effect on the overall kinematics of the mobile robot. The choice of wheel types for a mobile robot is strongly linked to the choice of wheel arrangement, or wheel geometry.

First of all there is the standard wheel as shown in Figure 2.1(a). The standard wheel has a roll axis parallel to the plane of the floor and can change orientation by rotating about an axis normal to the ground through the contact point. The standard wheel has two DOF. The caster offset standard wheel, also know as the castor wheel shown in Figure 2.1(b), has a rotational link with a vertical steer axis skew to the roll axis. The key difference between the fixed wheel and the castor wheel is that the fixed wheel can accomplish a steering motion with no side effects, as the center of rotation passes through the contact patch with the ground, whereas the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering [30].

(a) Standard Wheel (b) Castor Wheel

Figure 2.1: The standard wheel and castor wheel

The second type of wheel is the omnidirectional wheel (Figure 2.2). The omnidirectional wheel has three DOF and functions as a normal wheel, but

provides low resistance along the direction perpendicular to the roller direction as well. The small rollers attached around the circumference of the wheel are passive and the wheel's primary axis serves as the only actively powered joint. The key advantage of this design is that, although the wheel rotation is powered only along one principal axis, the wheel can kinematically move with very little friction along many possible trajectories, not just forward and backward.

Figure 2.2: Omniwheels

The third type of wheel is the ball or spherical wheel in Figure 2.3. It has also three DOF. The spherical wheel is a truly omnidirectional wheel, often designed so that it may be actively powered to spin along any direction. There have not been many attempts to build a mobile robot with ball wheels because of the difficulties in confining and powering a sphere. One mechanism for implementing this spherical design imitates the first computer mouse, providing actively powered rollers that rest against the top surface of the sphere and impart rotational force.

Figure 2.3: Spherical Wheels

The wheel type and wheel configuration are of tremendous importance,

they form an inseparable relation and they influence three fundamental characteristics of a: maneuverability, controllability, and stability. In general, there is an inverse correlation between controllability and maneuverability.

The number of wheels is the first decision. Two, three and four wheels are the most commonly used each one with different advantages and disadvantages. The two wheels drive has very simple control but reduced maneuverability. The three wheels drive has simple control and steering but limited traction. The four wheels drive has more complex mechanics and control, but higher traction [38].

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The motion vector of the robot is the sum of the independent wheel motions. The drive wheels are usually placed on each side of the robot. A non driven wheel, often a castor wheel, forms a tripod-like support structure for the body of the robot. Unfortunately, castors can cause problems if the robot reverses its direction. The castor wheel must turn half a circle and, the offset swivel can impart an undesired motion vector to the robot. This may result in to a translation heading error. Straight line motion is accomplished by turning the drive wheels at the same rate in the same direction. In place rotation is done by turning the drive wheels at the same rate in the opposite direction. Arbitrary motion paths can be implemented by dynamically modifying the angular velocity and/or direction of the drive wheels. The benefits of this wheel configuration is its simplicity.

A differential drive system needs only two motors, one for each drive wheel. Often the wheel is directly connected to the motor with internal gear reduction. Despite its simplicity, the controllability is rather difficult, especially to make a differential drive robot move in a straight line. Since the drive wheels are independent, if they are not turning at exactly the same rate the robot will veer to one side. Making the drive motors turn at the same rate is a challenge due to slight differences in the motors, friction differences in the drive trains, and friction differences in the wheel-ground interface. To ensure that the robot is traveling in a straight line, it may be necessary to adjust the motor speed very often. It is also very important to have accurate information on wheel position. This usually comes from the encoders. A round shaped differential drive configuration is shown in Figure 2.4.

In a tricycle vehicle (Figure 2.5) there are two fixed wheels mounted

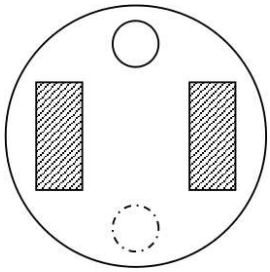

Figure 2.4: Differential drive configuration with two drive wheels and a castor wheel

on a rear axle and a steerable wheel in front. The fixed wheels are driven by a single motor which controls their traction, while the steerable wheel is driven by another motor which changes its orientation, acting then as a steering device. Alternatively, the two rear wheels may be passive and the front wheel may provide traction as well as steering.

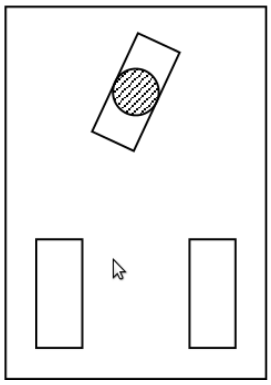

Figure 2.5: Tri-cycle drive, combined steering and driving

Another three wheel configuration is the synchro drive. The synchro drive system is a two motor drive configuration where one motor rotates all wheels together to produce motion and the other motor turns all wheels to change direction. Using separate motors for translation and wheel rotation guarantees straight line translation when the rotation is not actuated. This mechanical guarantee of straight line motion is a big advantage over the differential drive method where two motors must be dynamically controlled to produce straight line motion. Arbitrary motion paths can be done by actuating both motors simultaneously. The mechanism which permits all wheels to be driven by one motor and turned by another motor is fairly

complex. Wheel alignment is critical in this drive system, if the wheels are not parallel, the robot will not translate in a straight line. Figure 2.6 shows MRV4 a robot with this drive mechanism.

Figure 2.6: MRV4 robot with synchro drive mechanism

The car type locomotion or Ackerman steering configuration is used in cars. The limited maneuverability of Ackerman steering has an important advantage: its directionality and steering geometry provide it with very good lateral stability in high-speed turns. The path planning is much more difficult. Note that the difficulty of planning the system is relative to the environment. On a highway, path planning is easy because the motion is mostly forward with no absolute movement in the direction for which there is no direct actuation. However, if the environment requires motion in the direction for which there is no direct actuation, path planning is very hard. Ackerman steering is characterized by a pair of driving wheels and a separate pair of steering wheels. A car type drive is one of the simplest locomotion systems in which separate motors control translation and turning this is a big advantage compared to the differential drive system. There is one condition: the turning mechanism must be precisely controlled. A small position error in the turning mechanism can cause large odometry errors. This simplicity in line motion is why this type of locomotion is popular for human driven vehicles.

Some robots are omnidirectional, meaning that they can move at any time in any direction along the ground plane (x, y) regardless of the orientation of the robot around its vertical axis. This level of maneuverability requires omnidirectional wheels which present manufacturing challenges. Omnidirectional movement is of great interest to complete maneuverability.

Omnidirectional robots that are able to move in any direction (x, y, θ) at any time are also holonomic. There are two possible omnidirectional configurations.

The first omnidirectional wheel configuration has three omniwheels, each actuated by one motor, and they are placed in an equilateral triangle as depicted in Figure 2.7. This concept provides excellent maneuverability and is simple in design, however, it is limited to flat surfaces and small loads, and it is quite difficult to find round wheels with high friction coefficients. In general, the ground clearance of robots with Swedish and spherical wheels is somewhat limited due to the mechanical constraints of constructing omnidirectional wheels.

Figure 2.7: Palm Pilot Robot with omniwheels

The second omnidirectional wheel configuration has four omniwheel each driven by a separate motor. By varying the direction of rotation and relative speeds of the four wheels, the robot can be moved along any trajectory in the plane and, even more impressively, can simultaneously spin around its vertical axis. For example, when all four wheels spin 'forward' the robot as a whole moves in a straight line forward. However, when one diagonal pair of wheels is spun in the same direction and the other diagonal pair is spun in the opposite direction, the robot moves laterally. These omnidirectional wheel arrangements are not minimal in terms of control motors. Even with all the benefits, few holonomic robots have been used by researchers because of the problems introduced by the complexity of the mechanical design and controllability.

In mobile robotics the terms omnidirectional, holonomic and non holonomic are often used, a discussion of their use will be helpful. Omnidirectional simply means the ability to move in any direction. Because of the planar nature of mobile robots, the operational space they occupy contains

only three dimensions which are most commonly thought of as the x, y global position of a point on the robot and the global orientation, θ, of the robot. A non holonomic mobile robot has the following properties:

- The robot configuration is described by more than three coordinates. Three values are needed to describe the location and orientation of the robot, while others are needed to describe the internal geometry.
- The robot has two DOF, or three DOF with singularities.

A holonomic mobile robot has the following properties:

- The robot configuration is described by three coordinates. The internal geometry does not appear in the kinematic equations of the robot, so it can be ignored.
- The robot has three DOF without singularities.
- The robot can instantly develop a force in an arbitrary combination of directions x, y, θ.
- The robot can instantly accelerate in an arbitrary combination of directions x, y, θ.

Non holonomic robots are most common because of their simple design and ease of control. By their nature, non holonomic mobile robots have fewer degrees of freedom than holonomic mobile robots. These few actuated degrees of freedom in non holonomic mobile robots are often either independently controllable or mechanically decoupled, further simplifying the low-level control of the robot. Since they have fewer degrees of freedom, there are certain motions they cannot perform. This creates difficult problems for motion planning and implementation of reactive behaviors.
However, holonomic offer full mobility with the same number of degrees of freedom as the environment. This makes path planning easier because there are no constraints that need to be integrated. Implementing reactive behaviors is easy because there are no constraints which limit the directions in which the robot can accelerate.

2.2 Motion Models

In the field of robotics the topic of robot motion has been studied in depth in the past. Robot motion models play an important role in modern robotic algorithms. The main goal of a motion model is to capture the relationship between a control input to the robot and a change in the robot's pose. Good models will capture not only systematic errors, such as a tendency of the robot to drift left or right when directed to move forward, but will also capture the stochastic nature of the motion. The same control inputs will almost never produce the same results and the effects of robot actions are, therefore, best described as distributions [41]. Borenstein et al. [32] cover a variety of drive models, including differential drive, the Ackerman drive, and synchro-drive.

Previous work in robot motion models have included work in automatic acquisition of motion models for mobile robots. Borenstein and Feng [31] describe a method for calibrating odometry to account for systematic errors. Roy and Thrun [41] propose a method which is more amenable to the problems of localization and SLAM. They treat the systematic errors in turning and movement as independent, and compute these errors for each time step by comparing the odometric readings with the pose estimate given by a localization method. Alternately, instead of merely learning two simple parameters for the motion model, Eliazat and Parr [15] seek to use a more general model which incorporates interdependence between motion terms, including the influence of turns on lateral movement, and vice-versa. Martinelli et al. [16] propose a method to estimate both systematic and non-systematic odometry error of a mobile robot by including the parameters characterizing the non-systematic error with the state to be estimated. While the majority of prior research has focused on formulating the pose estimation problem in the Cartesian space. Aidala and Hammel [29], among others, have also explored the use of modified polar coordinates to solve the relative bearing-only tracking problem. Funiak et al. [40] propose an over-parameterized version of the polar parameterization for the problem of target tracking with unknown camera locations. Djugash et al. [24] further extend this parameterization to deal with range-only measurements and multimodal distributions and further extend this parameterization to improve the accuracy of estimating the uncertainty in the motion rather than the measurement.

2.3 Navigation

A navigation environment is in general dynamic. Navigation of autonomous mobile robots in an unknown and unpredictable environment is a challenging task compared to the path planning in a regular and static terrain, because it exhibits a number of distinctive features. Environments can be classified as known environments, when the motion can be planned beforehand, or partially known environments, when there are uncertainties that call for a certain type of on-line planning for the trajectories. When the robot navigates from original configuration to goal configuration through unknown environment without any prior description of the environment, it obtains workspace information locally while it is moving and a path must be incrementally computed as the newer parts of the environment are explored [26]. Autonomous navigation is associated to the capability of capturing information from the surrounding environment through sensors, such as vision, distance or proximity sensors. Even though the fact that distance sensors, such as ultrasonic and laser sensors, are the most commonly used ones, vision sensors are becoming widely applied because of its ever-growing capability to capture information at low cost.

Visual control methods fall into three categories such as position based, image based and hybrid [28]. The position based visual control method reconstructs the object in 3D space from 2D image space, and then computes the errors in Cartesian space. For example, Han et al [25] presented a position based control method to open a door with a mobile manipulator, which calculated the errors between the end-effector and the doorknob in Cartesian space using special rectangle marks attached on the end-effector and doorknob. As Hager [28] pointed out, the position based control method has the disadvantage of low precision in positioning and control. To improve the precision, El-Hakim et al [39] proposed a visual positioning method with 8 cameras, in which the positioning accuracy was increased through iteration. It has high positioning accuracy but poor performance in real time.

The image based visual control method does not need to reconstruct in 3D space, but the image Jacobian matrix needs to be estimated. The controller design is difficult. And the singular problem in image Jacobian matrix limits its application [28]. Hybrid control method attempts to give a good solution through the combination of position and image based visual control methods. It controls the pose with position based method, and the position with image based method. For example, Malis et al [35] provided

a 2.5 D visual control method. Deguchi et al. [22] proposed a decoupling method of translation and rotation. Camera calibration is a tedious task, and pre-calibration cameras used in visual control methods limit a lot the flexibility of the system. Therefore, many researchers pursue the visual control methods with self-calibrated or un-calibrated cameras. Kragic et al. [34] gave an example to self-calibrate a camera with the image and the CAD model of the object in their visual control system. Many researchers proposed various visual control methods with un-calibrated cameras, which belong to image based visual control methods. The camera parameters are not estimated individually, but combined into the image Jacobian matrix. For instance, Shen et al. [43] limited the working space of the end-effector on a plane vertical to the optical axis of the camera to eliminate the camera parameters in the image Jacobian matrix. Xu et al. [21] developed visual control method for the end-effector of the robot with two un-calibrated cameras, estimating the distances based on cross ratio invariance.

2.4 Games and Interaction

Advances in the technological medium of video games have recently included the deployment of physical activity-based controller technologies, such as the Wii [27], and vision-based controller systems, such as Intel's Me2Cam [13]. The rapid deployment of millions of iRobot Roomba home robots [14] and the great popularity of robotic play systems, such as LEGO Mindstorms and NXT [5] now present an opportunity to extend the realm of video game even further, into physical environments, through the direct integration of human-robot interaction techniques and architectures with video game experiences.

Over the past thirty to forty years, a synergistic evolution of robotic and video game-like programming environments, such as Turtle Logo [36], has occurred. At the MIT Media Lab, these platforms have been advanced through the constructionist pedagogies, research, and collaborations of Seymour Papert, Marvin Minsky, Mitch Resnick, and their colleagues, leading to Logo [7], Star Logo [37], programmable Crickets and Scratch [6] and Lego MindStorms [37]. In 2000, Kids Room [18] demonstrated that an immersive educational gaming environment with projected objects and characters in physical spaces (e.g., on the floor or walls), could involve children in highly interactive games, such as hide-and-seek. In 2004, RoBallet [20] advanced these constructionist activities further, blending elements of projected vir-

tual environments with sensor systems that reacted to children dancing in a mediated physical environment. The realm of toys and robotic pets has also seen the development of a wide array of interactive technologies (e.g., Furby, Aibo, Tamagotchi) and more recently Microsoft's Barney [9], which has been integrated with TV-based video content. Interactive robotic environments for education are now being extended to on-line environments, such as CMU's educational Mars rover [8], and becoming popular through robotics challenges such as FIRST Robotics Competition [3], BattleBots [1], and Robot World Cup soccer tournaments, such as Robocup [42].

The games related with robots, so called robogames, are categorized into four branches according to AIRLab report [19]. One is the videogames, where robot characters are simulated. Soccer Simulation League in RoboCup-Soccer is an example of this kind of games. The Simulation League focuses on artificial intelligence and team strategy. Independently moving software players (agents) play soccer on a virtual field inside a computer. This provides a context to the game, but also allows to escape all the limitations of physical robots. Another one is the tele-operated physical robots, where the player is mainly in the manipulation of remote controllers similar to the ones used in videogames, or, eventually, in the physical building of the tele-operated robots, as it happens with RoboWars [11]. A third main stream concerns robots that have been developed by roboticists to autonomously play games (e.g., Robocup). Here, the accent is on the ability to program the robots to be autonomous, but little effort is spent in the eventual playful interaction with people, often avoided, as in most of the Robocup leagues. The last main stream concerns robots that act as more or less like mobile pets. In this case, interaction is often limited to almost static positions, not exploiting rich movement, nor high autonomy; the credibility of these toys to really engage healthy people, such as kids, is not high.

According to the AIRLab report [19], a new category of games where the players are involved in a challenging and highly interactive game activity with autonomous robots called as Highly Interactive, Competitive RoboGames (HI-CoRG) is introduced. The idea is to take the videogame players away from screen and console, and to make them physically interact with a robot in their living environment. In this context some heuristics from videogames adapted to be applied on this HI-CoRG games.

In our thesis, we focused on developing a robot for games that can be count to the HI-CoRG category.

We introduce now some of the robots developed in the past, related to the human robot interaction and games.

Kismet (Figure 2.8(a)) is a robot made in the late 1990s at Massachusetts Institute of Technology with auditory, visual and expressive systems intended to participate in human social interaction and to demonstrate simulated human emotion and appearance. This project focuses not on robot-robot interactions, but rather on the construction of robots that engage in meaningful social exchanges with humans. By doing so, it is possible to have a socially sophisticated human assist the robot in acquiring more sophisticated communication skills and helping it to learn the meaning of these social exchanges.

A Furby (Figure 2.8(b)) was a popular electronic robotic toy resembling a hamster/owl-like creature which went through in 1998. Furbies were the first successful attempt to produce and sell a domestically-aimed robot. A newly purchased Furby starts out speaking entirely Furbish, the unique language that all Furbies use, but are programmed to speak less Furbish as they gradually start using English. English is learned automatically, and no matter what culture they are nurtured in, they learn English. In 2005, new Furbies were released, with voice-recognition and more complex facial movements, and many other changes and improvements.

AIBO (Artificial Intelligence Robot) (Figure 2.8(c)) was one of several types of robotic pets designed and manufactured by Sony. There have been several different models since their introduction on May 11, 1999. AIBO is able to walk, "see" its environment via camera and recognize spoken commands in Spanish and English. AIBO robotic pets are considered to be autonomous robots since they are able to learn and mature based on external stimuli from their owner, their environment and from other AIBOs. The AIBO has seen use as an inexpensive platform for artificial intelligence research, because it integrates a computer, vision system, and articulators in a package vastly cheaper than conventional research robots. The RoboCup autonomous soccer competition had a "RoboCup Four-Legged Robot Soccer League" in which numerous institutions from around the world would participated. Competitors would program a team of AIBO robots to play games of autonomous robot soccer against other competing teams.

The developments in Robocup lead to improvements in the mobile robots.

(a) Kismet (b) Furby (c) AIBO

(d) Spykee (e) Rovio

Figure 2.8: Robots developed for games and interactions

The domestically-aimed robots become popular in the market. One of them is Spykee (Figure 2.8(d)), which is a robotic toy made by Meccano in 2008. It contains a USB webcam, microphone and speakers. Controlled by computer locally or over the internet, the owner can move the robot to various locations within range of the local router, take pictures and video, listen to surroundings with the on-board microphone and play sounds/music or various built-in recordings (Robot laugh, laser guns, etc.) through the speaker. Spykee has a WiFi connectivity to let him access the Internet using both ad-hoc and infrastructure modes.

Similar to Spykee, with different kinematics models and more improvements, Rovio^{TM}(Figure 2.8(e)) is the groundbreaking new Wi-Fi enabled mobile webcam that views and interacts with its environment through video and audio streaming.

According to our goals, we investigated the previously made robots, since we thought we could benefit from the techniques used. Furbies have expressions, but they don't move. While Rovio has omnidirectional, Spykee has tracks for the motion, but they lack entertainment. AIBO has legs and a

lot of motors, but these brings more cost and high complexity for the development.

Chapter 3

Mechanical Construction

We started our design from these specifications of the robot.

- a dimension of about 25 cm of radius, 20 cm height
- a speed of about 1 m/sec
- sensors to avoid obstacles
- a camera that can be moved up and down
- power enough to move and transmit for at least 2 hours without recharging
- the robot should cost no more than 250 euro.

In the development process we faced some problems due to the limitations from the specifications. Main causes of these problems are related with low-cost, that is coming with our design constraints.

The mechanical construction of the robot is focused on construction of the robot chassis, motor holders, motor and wheel connections, camera holder, the foam covering the robot, batteries and hardware architectures.

3.1 Chassis

The main principles for the construction of the chassis are coming from similar projects from the past, which are the simplicity of assembly and disassembly, the ease of access to the interior and the possibility of adding

and modifying elements in the future. We decided to use some design constraints, revising these according to our goals.

The design is started with the choice of the chassis made of plexiglas. One advantage of using plexiglas, it is 43% lighter than aluminum [10]. Another advantage that is affecting our choice is the electrical resistance of the plexiglas, that will isolate any accidental short circuit. One of the major problems with plexiglas is the difficult processing of the material. However, it has to be processed only once, hence this is negligible.

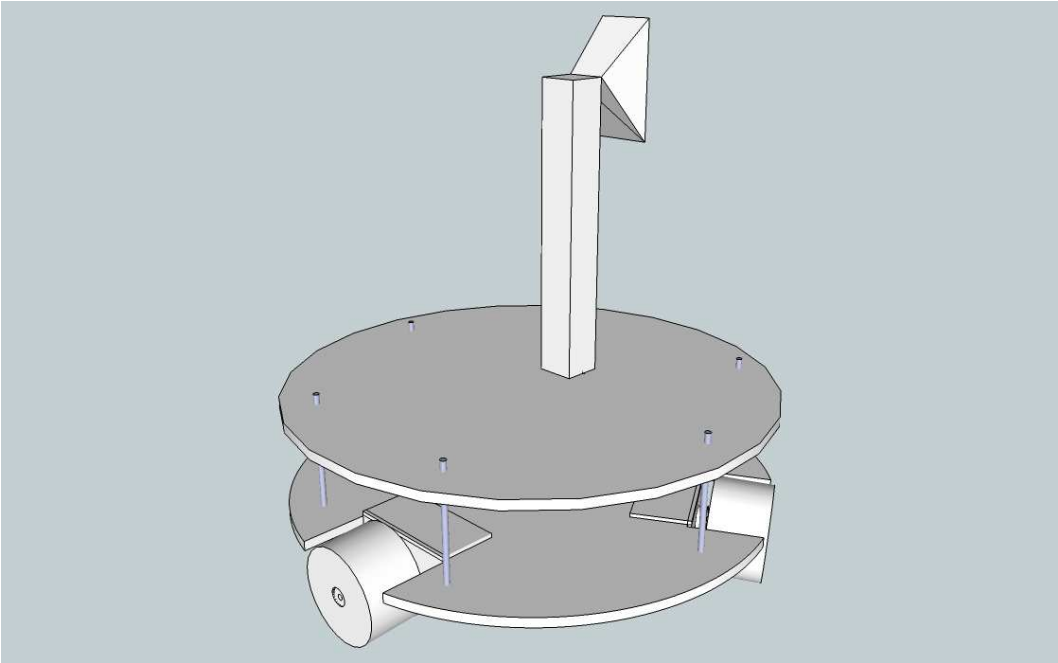

Figure 3.1: The design of robot using Google SketchUp

The preliminary design has been created with Google SketchUp, allowing to define the dimensions of the robot and the arrangement of various elements shown in Figure 3.1. This model has been used to obtain a description of the dynamics of the system. The robot is 125 mm in diameter wide and 40 cm in height, meeting the specification to be contained in a footprint on the ground in order to move with agility in the home. The space between the two plates is around 6 cm, which allows us to mount sensors and any device that will be added in the future. The total weight of the structure is approximately 1.8 kg, including motors and batteries.

The initial design of the chassis was a bit different from the final configuration seen in Figure 3.1. Even though the shape of the components did not change, the position and orientation are changed in the final configuration. The motor holders initially were intended to be placed on the top of the bottom plexiglas layer. At the time when this decision was taken, we were not planning to place the mice boards, but only to put the batteries and the motor control boards. Later, with the decision of placing the mice boards in this layer, in order to get more space, we decided to put the motors to their final position. So this configuration increases the free space on the robot layers to put the components, and also increases the robot height from the ground that will result to better navigation.

Another change has been made by placing the second plexiglas layer. Initially, we placed that layer using only three screws with each a height of 6 cm. The idea was using minimum screws, so that the final weight will be lighter and the plexiglas will be more resistant to damage. Later, when we placed the batteries, motor controller boards and the camera with its holder, the total weight was too much to be handled by the three screws. And additionally, we placed 6 more screws with the same height as before. These screws, allowed us to divide the total weight on the plate equally on all the screws and also enabled us to install springs and foams, to implement bumpers that protect the robot from any damage that could be caused by hits.

3.2 Motors

The actuator is one of the key components in the robot. Among the possible actuation we decided to go with DC motors. Servo motors are not powerful enough to reach the maximum speed. Due to noise and control circuitry requirements, servos are less efficient than uncontrolled DC motors. The control circuitry typically drains 5-8mA on idle. Secondly, noise can draw more than triple current during a holding position (not moving), and almost double current during rotation. Noise is often a major source of servo inefficiency and therefore they should be avoided. Brushless motors are more power efficient, have a significantly reduced electrical noise, and last much longer. However, they also have several disadvantages, such as higher price and the requirement for a special brushless motor driver. Since they are running at high speed we need to gear them down. This would also add

some extra cost. Also the Electronic Speed Controllers(ESC) are costly and most of them do not support multiple run motors. The minimum price for the motor is about 20 dollars, and for the ESC is around 30 dollars. The minimum expected price for motors and controller will be a least 90 dollars if we can run the 3 motors on a single controller. Also there should be an extra cost to gear them down.

We made some calculations to find the most suitable DC motor for our system. In order to effectively design with DC motors, it is necessary to understand their characteristic curves. For every motor, there is a specific torque/speed curve and power curve. The graph in Figure 3.2 shows a torque/speed curve of a typical DC motor.

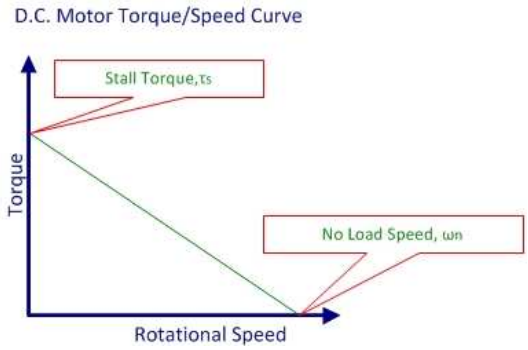

Figure 3.2: The optimal torque / speed curve

Note that, torque is inversely proportional to the speed of the output shaft. In other words, there is a trade-off between how much torque a motor delivers, and how fast the output shaft spins. Motor characteristics are frequently given as two points:

- The stall torque, τ_s, represents the point on the graph at which the torque is a maximum, but the shaft is not rotating.
- The no load speed, ω_n, is the maximum output speed of the motor (when no torque is applied to the output shaft).

The linear model of a DC motor torque/speed curve is a very good approximation. The torque/speed curves shown below in Figure 3.3 are calculated curves for our motor, which is Pololu 25:1 Metal Gearmotor 20Dx44L mm.

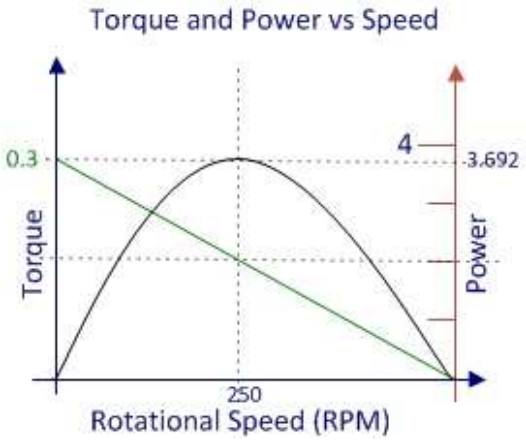

Figure 3.3: The calculated curve for the Pololu 25:1 Metal Gearmotor 20Dx44L mm

Due to the linear inverse relationship between torque and speed, the maximum power occurs at the point where $\omega = \frac{1}{2}\omega_n$, and $\tau = \frac{1}{2}\tau_s$. The maximum power output occuring at no load speed with, $\tau = 500rpm = 52.38rad/sec$, and the stall torque, $\omega = 0.282Nm$ is calculated as follows:

$$P = \tau * \omega$$

$$P_{\max} = \frac{1}{2}\tau_s * \frac{1}{2}\omega_n$$

$$P_{\max} = 26.190rad/sec * 0.141Nm = 3.692W$$

Keeping in mind the battery life, the power consumption, the necessary torque and the maximum speed, we selected the Pololu motors shown in Figure 3.4.

In the design of the robot we decided to use low cost components. In that sense we focused on producing components or re-using components that can be modified according to our demands. The mechanical production of the components took some time both for the design and the construction process (e.g. the connectors between motors and wheels are milled from an aluminum bar), however this reduced the overall cost. The connection of motors with robot is made by putting the motor inside an aluminum tube, merging it with the U-shaped plate (Figure 3.5). Using such a setup helps

Figure 3.4: Pololu 25:1 Metal Gearmotor 20Dx44L mm. Key specs at 6 V: 500 RPM and 250 mA free-run, 20 oz-in (1.5 kg-cm) and 3.3 A stall.

not only protecting the motor from the hit damage, but also cooling of the motor since aluminum has great energy-absorbing characteristics [12]. The connection can be seen clearly in Figure 3.5. The component is attached to the plexiglas from the L-shaped part using a single screw. This gives us the flexibility to dis-attach the component easily and to change the orientations of them if needed.

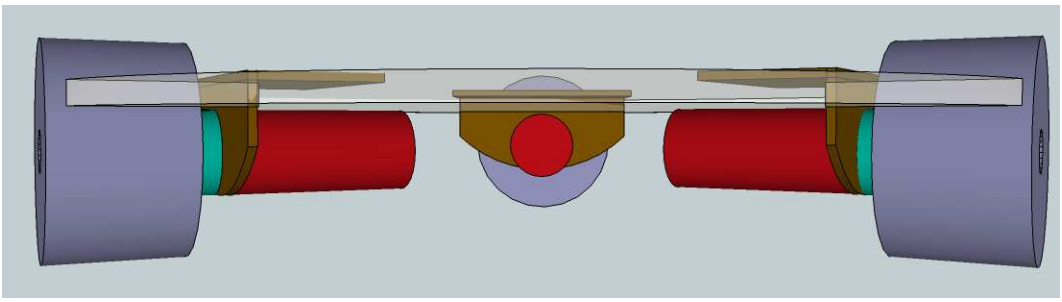

Figure 3.5: The connection between the motors, chassis and wheels

3.3 Wheels

The target environment of our robot is a standard home. The characteristic properties of this environment that are important for our work are as follows. Mainly the environment is formed by planes, surfaces such as parquet, tile, carpet etc... In order to move freely to any direction on these surfaces and reach the predefined speed constraint, we selected the wheels and a proper configuration for them. The decision to choose omnidirectional wheel was motivated, but there are lots of different omnidirectional wheels available on the market. Among them, we made a selection considering the

target surface, maximum payload, weights of the wheels, and price. The first selected wheel was the omniwheel shown in Figure 3.6(a). The wheel consists of three small rollers, which may affect the turning since the coverage is not good enough. Also the wheel itself is heavier than the one with the transwheel shown in Figure 3.6(b) A single transwheel is 0.5 oz lighter than an omniwheel. Another model is the double transwheel seen in Figure 3.6(c), which is produced by merging two transwheels, where the rollers are covering all the wheel, which will enable the movement in any direction easily and more consisting model by reducing the possible power transmission loss that can be occur, when merging the two wheels by hand.

(a) Omniwheel (b) Transwheel (c) Double Transwheel

Figure 3.6: Wheels

In order reach the maximum speed of 1 m/sec., we should have the following equation.

$$speed = circumreference * rps$$

$$speed = diameter * pi * rps$$

As it can be seen from the equation the speed is also related with the rotation per second (rps) of the wheels, which is determined by the motor. So the dimension choice of the wheels are made keeping the rps in mind. The rpm necessary to turn our wheels with the maximum speed of 1 meter/second the is calculated as follows:

$$1000mm/second = diameter * pi * rps$$

$$1000mm/second = 49.2mm * pi * rps$$

$$rps \cong 6.4$$

$$rpm \cong 388$$

As a result of the calculation, using the omniwheel with outer diameter of 49.2 m., we will need a motor that can run around 388 rpm to reach the maximum speed.

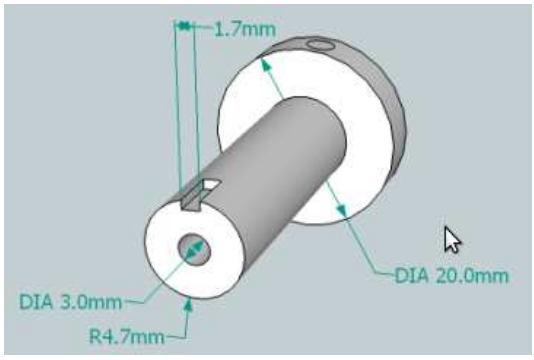

Figure 3.7: The wheel holder bar, that is making the transmission between motors and wheels

The transmission between motors and wheels is achieved by the bar, which is lathed from an aluminum bar (Figure 3.7). The bar is placed inside the wheel and locked with a key using the key-ways in the wheel and the bar.

For the wheel configuration we preserved the popular three wheeled configuration (Figure 3.8). The control is simple, the maneuverability is enough to satisfy the design specifications. The details of this configuration will be mentioned in Control Chapter.

3.4 Camera

The camera positioning is tricky. We needed a holder that should be light in the weight, but also provide enough height and width to enable vision from the boundary of the robot at ground to the people face in the environment. The initial tests have been made by introducing a camera holder using parts from ITEM [4]. These parts are useful during the tests since they are easily configurable for different orientations, and easy to assemble. But, the parts are too heavy and we decided to use an aluminum bar for the final configuration. The movement of the camera is done by the servo placed at the top of the aluminum bar; this gave us the flexibility to have different camera

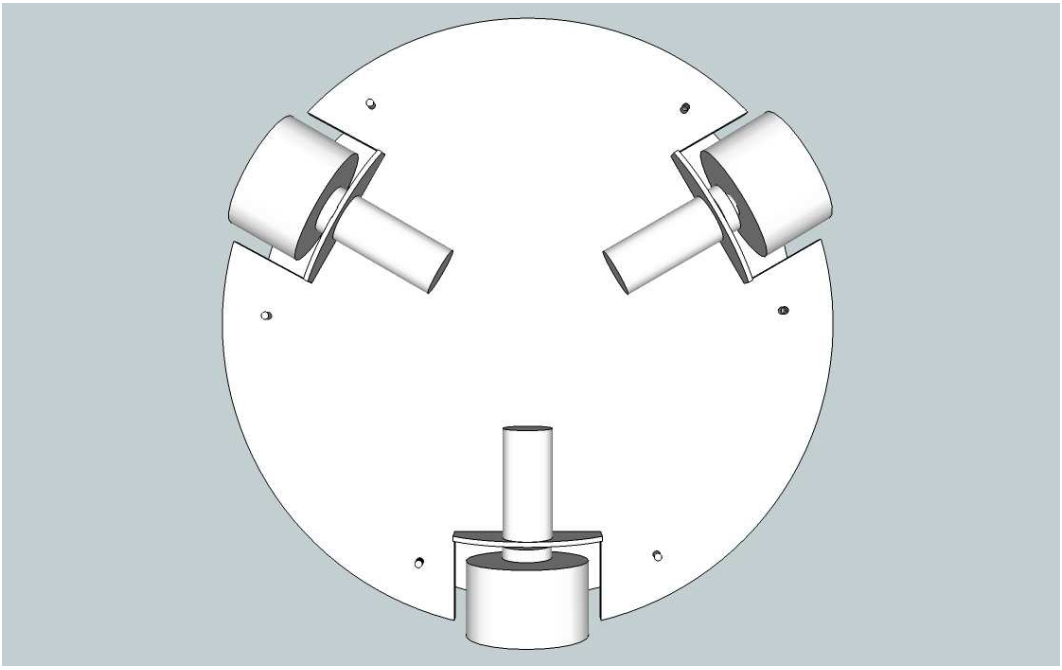

Figure 3.8: Three wheel configuration

positions, that will be useful to develop different games.

The camera is placed in a position on top of a mounted-on aluminum bar that allows us to have the best depth of field by increasing the field of view. The idea is to detect objects and visualize the environment between the boundary of the robot to all the way in the ground and up to 2 meters in height, which allows also to see faces of people or the objects not on the ground. Mechanically, the camera itself is connected to a servo that enables the camera head to move freely in 120°in the vertical axis, as shown in Figure 3.9 and 3.10. This configuration gives us the flexibility to generate different types of games using the vision available in a wide angle and interchangeable height.

3.5 Bumpers

The last mechanical component is the collision detection mechanism, to avoid obstacles in the environment. There are lots of good solutions to this issue. By using different types of sensors such as sonars, photo resistors,

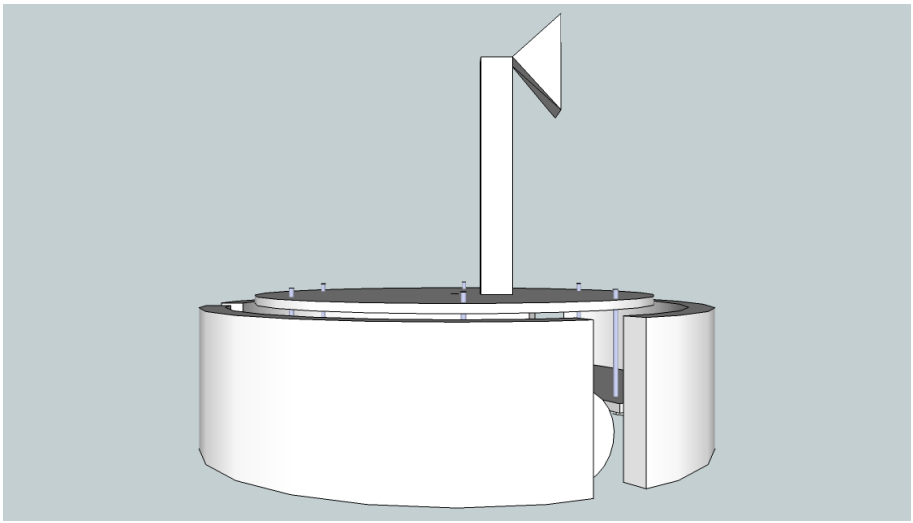

Figure 3.9: The position of the camera

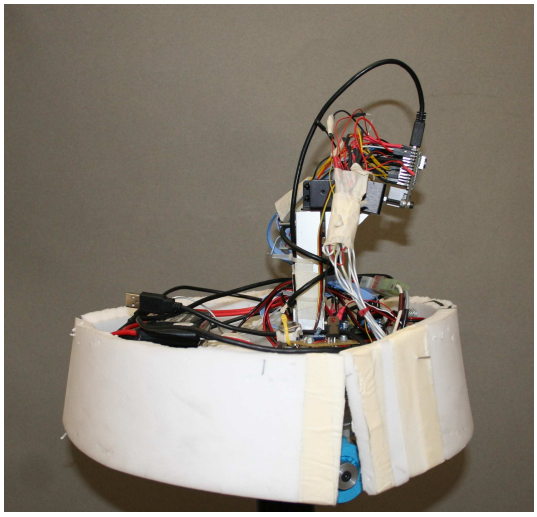

Figure 3.10: The real camera position

IR sensors, tactile bumpers, etc. Among them, the simplest are the tactile bumpers. A tactile bumper is probably one of the easiest way of letting a robot know if it's colliding with something. Indeed, they are implemented by electrical switches. The simplest way to do this is to fix a micro switch to robot in a way so that when it collides the switch will be pushed in, making an electrical connection. Normally the switch will be held open by an internal spring. Tactile bumpers are great for collision detection, but the circuit

itself also works fine for user buttons and switches as well. There are many designs possible for bump switches, often depending on the design goals of the robot itself. But the circuit remains the same. They usually implement a mechanical button to short the circuit, pulling the signal line high or low. An example is the micro switch with a lever attached to increase its range, as shown in Figure 3.11. The cost is nothing if compared to the other solutions such as photo-resistors and sonars, and the usage is pretty simple since the values can be read directly from the microcontroller pins without having any control circuits. Major drawback is its limited range, but we tried to improve the range using the foam and the external springs attached to the foam. Since the robot is light in the weight and collision can be useful in development of games, we decided to use tactile bumpers.

Figure 3.11: Bumpers are mechanical buttons to short the circuit, pulling the signal line high or low.

The collision detection for robot is made with bumpers, which are placed on the plexiglas every 60°(Figure 3.12). The coverage was not enough, so the bumpers are covered with foams which are connected to the springs. The springs are enabling the push back of the switches, the foams are increasing the coverage of the collision detection and also enhance the safety both for the damage that could be caused by the robot and to the robot from environment (Figure 3.13). After some tests we realized there are still dead points which the collision are not detected. We decided to cut the foam into three, placing the around the robot leaving the parts with the wheel open. The best results are obtained using this configuration so we decided to keep it.

3.6 Batteries

The robot's battery life without the need of recharging is crucial for the game. The game play must continue for about 2 hours without any interruption. This brings the question of how to choose the correct battery. LiPo batteries are suitable battery choice for our application over conven-

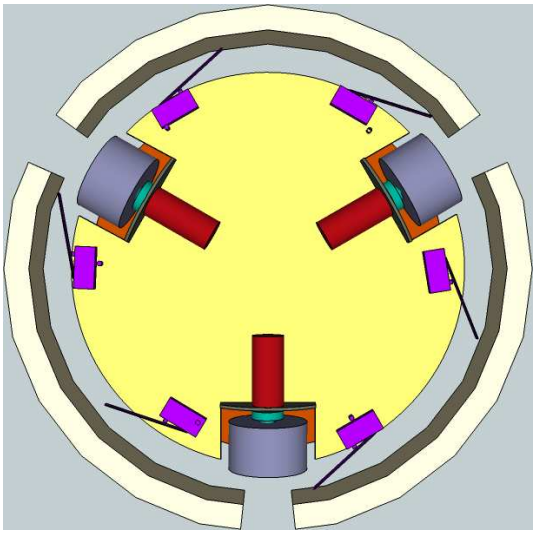

Figure 3.12: The bumper design

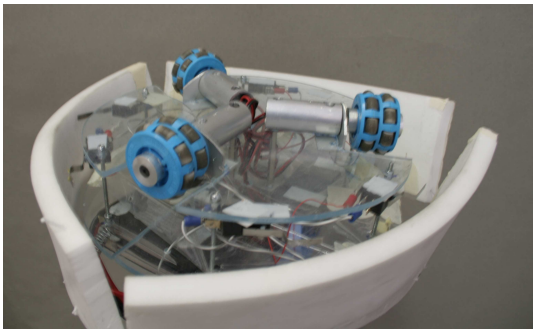

Figure 3.13: Robot with foams, springs and bumpers

tional rechargeable battery types such as NiCad, or NiMH, for the following reasons :

- LiPo batteries are light in weight and can be made in almost any shape and size.
- LiPo batteries have large capacities, meaning they hold lots of power in a small package.
- LiPo batteries have high discharge rates to power the most demanding electric motors.

In short, LiPo provide high energy storage to weight ratios in an endless variety of shapes and sizes. The calculation is made to find the correct battery. The motors are consuming 250 mA at free-run and 3300 mA for the stall current. For the all three motors we should have the following battery lives:

$$Battery\_Capacity/Current\_Draw = Battery\_Life$$

$$2 * 2500mAh/750mA \cong 6.6hours$$

$$2 * 2500mAh/9900mA \cong 0.5hours$$

using the 2 batteries each having a capacity of 2500 mAh. The battery life shows changes according to the current draw of the motors. In case, each motor is consuming 250 mA in free-run current will result 6.6 hours of batteries life. On the other hand, with the stall current it will be 0.5 hour battery life. Since the motor will not always work in stall current or the free-run current; the choice of 2500 mA batteries (Figure 3.14) seems to be enough to power the robot for at least 2 hours.

Figure 3.14: J.tronik - Battery Li-Po Li-POWER 2500 mA 2S1P 7,4V 20C

3.7 Hardware Architecture

During the development of the robot, we used several hardware pieces such as microprocessor, camera, motor control boards, voltage regulator circuit, voltage divider circuit. Most of them were already developed systems and we did not focus on the production details of them. We only created the voltage regulator and divider circuit, which we used in order to power the boards and measure the battery level of charge.

The main component in our system is the the STL Main Board, known also as STLCam. The STL Main Board is a low-cost vision system for acquisition and real-time processing of pictures, consisting of a ST-VS6724 Camera (2 Mpx), a ST-STR912FA Microcontroller (ARM966 @ 96MHz) and 16MB of external RAM (PSRAM BURST). The schematics of the STL Main Board is shown in Appendix D.2.

ST-STR912FAZ44 Microcontroller

The microcontroller main components are: a 32 bit ARM966E-S RISC processor core running at 96MHz, a large 32bit SRAM (96KB) and a high-speed 544KB Flash memory. The ARM966E-S core can perform single-cycle DSP instructions, good for speech recognition, audio and embedded vision algorithms.

ST-VS6724 Camera Module

The VS6724 is a UXGA resolution CMOS imaging device designed for low power systems, particularly mobile phone and PDA applications. Manufactured using ST 0.18μ CMOS Imaging process, it integrates a high-sensitivity pixel array, digital image processor and camera control functions. The device contains an embedded video processor and delivers fully color processed images at up to 30 fps UXGA JPEG, or up to 30 fps SVGA YCbCr 4:2:2. The video data is output over an 8-bit parallel bus in JPEG (4:2:2 or 4:2:0), RGB, YCbCr or Bayer formats and the device is controlled via an I2C interface. The VS6724 camera module uses ST's second generation SmOP2 packaging technology: the sensor, lens and passives are assembled, tested and focused in a fully automated process, allowing high volume and low cost production. The VS6724 also includes a wide range of image enhancement functions, designed to ensure high image quality, these include: automatic exposure control, automatic white balance, lens shading compensation, defect correction algorithms, interpolation (Bayer to RGB conversion), color space conversion, sharpening, gamma correction, flicker cancellation, NoRA noise reduction algorithm, intelligent image scaling, special effects.

MC33887 Motor Driver Carrier

All electric motors need some sort of controller. The motor controller may different features and complexity depending on the task that the motors will have to perform.

The simplest case is a switch to connect a motor to a power source, such as in small appliances or power tools. The switch may be manually operated or may be a relay or conductor connected to some form of sensor to automatically start and stop the motor. The switch may have several positions to select different connections of the motor. This may allow reduced-voltage starting of the motor, reversing control or selection of multiple speeds. Overload and over-current protection may be omitted in very small motor controllers, which rely on the supplying circuit to have over-current protection.

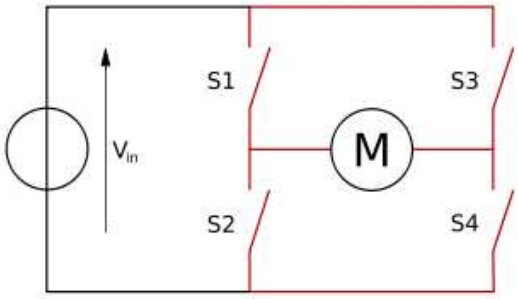

Figure 3.15: Structure of an H bridge (highlighted in red)

The DC motors cannot be controlled directly from the output pins of the microcontroller. We need the circuit so called 'motor controller', 'motor driver' or an 'H-Bridge'. The term H-Bridge is derived from the typical graphical representation of such a circuit. An H-Bridge (Figure 3.15) is built with four switches (solid-state or mechanical). When the switches S1 and S4 are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

To drive motors we used a PWM signal and vary the duty cycle to act as a throttle: 100% duty cycle = full speed, 0% duty cycle = coast, 50% duty cycle = half speed etc. After some testing we optimized the percentage of the duty cycle in order achieve a better performance. This optimization will be mentioned later in Control Chapter.

For the motor control, we started by using the H-Bridge motor control circuits provided by our sponsor. The initial tests have been performed by implementing the correct PWM waves using these boards. Later, we real-

ized that the boards were configured to work at 8 V. This forced us to make the decision of buying new batteries or new control circuits. Evaluating the prices, we ended up buying new control circuits that are rated for 5 V.

MC33887 motor driver integrated circuit is an easy solution to connect a brushed DC motor running from 5 to 28 V and drawing up to 5 A (peak). The board incorporates all the components of the typical application, plus motor-direction LEDs and a FET for reverse battery protection. A microcontroller or other control circuit is necessary to turn the H-Bridge on and off. The power connections are made on one end of the board, and the control connections (5V logic) are made on the other end. The enable (EN) pin does not have a pull-up resistor, so it be must pulled to +5 V in order to wake the chip from sleep mode. The fault-status (FS, active low) output pin may be left disconnected if it is not needed to monitor the fault conditions of the motor driver; if it is connected, it must use an external pull-up resistor to pull the line high. IN1 and IN2 control the direction of the motor, and D2 can be PWMed to control the motor's speed. D2 is the "not disabled" line: it disables the motor driver when it is driven low (another way to think of it is that, it enables the motor driver when driven high). Whenever D1 or D2 disable the motor driver, the FS pin will be driven low. The feedback (FB) pin outputs a voltage proportional to the H-Bridge high-side current, providing approximately 0.59 volts per amp of output current.

Voltage Divider and Voltage Regulator Circuit

Batteries are never at a constant voltage. For our case 7.2 V battery will be at around 8.4 V when fully charged, and can drop to 5 V when drained. In order to power microcontroller (and especially sensors) which are sensitive to the input voltage, and rated to 5 V, we need a voltage regulator circuit to output always 5 V. The design of the circuit that will be used in voltage regulation merged with the voltage divider circuit that will be used for battery charge monitor shown in Figure 3.16.

To operate voltage divider circuit, the following equation is used to determine the appropriate resistor values.

$$V_o = \frac{V_i}{R_1 + R_2} * R_2$$

V_i is the input voltage, R_1 and R_2 are the resistors, and V_o is the output voltage.

With the appropriate selection of resistor R_1 and R_2 based on the above information, V_o will be suitable for the analog port on microcontroller. The divider is used to input to the microcontroller a signal proportional to the voltage provided by the battery, so to check its charge. Note that a fully charged battery can often be up to 20% more of its rated value and a fully discharged battery 20% below its rated value. For example, a 7.2 V battery fully charged can be 8.4 V, and fully discharged 5 V. The voltage divider circuit allows to read the battery level from the microcontroller pins directly, that will be used in order to monitor battery charging level changes.

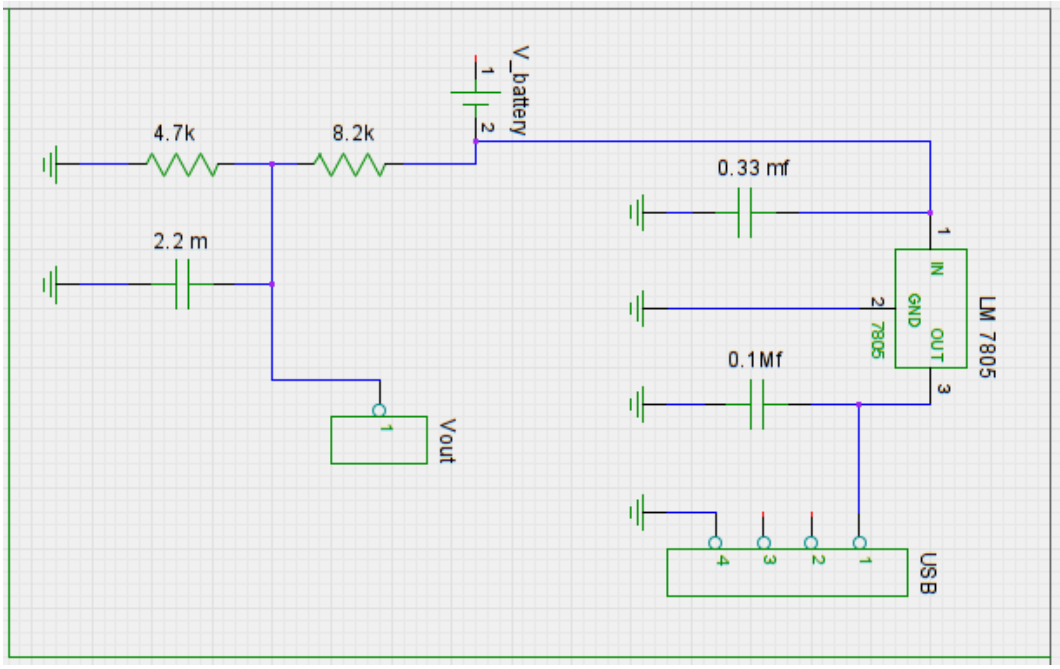

Figure 3.16: The schematics of voltage divider and voltage regulator circuit

From the Figure 3.16, the pin V_{out} is used in order to monitor the battery life, which is placed on the left side of schematics. The divided voltage is read from the analog input of the microcontroller. The voltage regulator circuit which is placed on the right part of the schematics is used to power the STLCam board with 5 V through the USB port. It can be also used in order to make the connection with the PC, to transmit some data, which we use for debug purposes.

Chapter 4

Control

The motion mechanism is inspired from the ones that have been introduced.

In this Chapter, we will explain the wheel configuration model, movement types, the script that is used to calculate the motor contributions according to a set point, motor control behavior and the software implementation.

4.1 Wheel configuration

The configuration space of an omnidirectional mobile robot, is a smooth 3-manifold and can then be locally embedded in Euclidean space R^3. The robot has three degrees of freedom, i.e., two dimension linear motion and one dimension rotational motion. There are three universal wheels mounted along the edge of the robot chassis 120°apart from each other, and each wheel has a set of rollers aligned with its rim, as shown in Figure 3.8. Because of its special mechanism, the robot is able to simultaneously rotate and translate. Therefore, the path planning can be significantly simplified by directly defining the tracking task with the orientation and position errors obtained by the visual feedback.

For a nonholonomic robot, the robot's velocity state is modeled as the motion around its instantaneous center of rotation (ICR). As a 2D point in the robot frame, the ICR position can be represented using two independent parameters. One can use either Cartesian or polar coordinates, but singularities arise when the robot moves in a straight line (the ICR thus lies at infinity). Hence, we used a hybrid approach that is defining the robot

position both in Cartesian and polar coordinates.

Normally, the position of the robot is represented by Cartesian coordinates which is a point in X-Y plane. By polar coordinates, the position is described by an angle, and a distance to the origin. Instead of representing robot position with a single coordinate, the hybrid approach is used as follows. The robot pose is defined as $\{X_I, Y_I, \alpha\}$ where X_I and Y_I are linear positions of the robot in the world. Let α denote the angle between the robot axis and the vector that connects the center of the robot and the target object.

The transformation of the coordinates into polar coordinates with its origin at goal position:

$$p = \sqrt{\Delta x^2 + \Delta y^2} \text{ and } \alpha = -\theta + atan2(\Delta x, \Delta y) \tag{4.1}$$

Then the calculated angle α is passed as the parameter to the simulator in order to test the motor contributions calculated for the motion. Later, the behavior tested with simulator is implemented for microcontroller with Triskar function (in Appendix A.1), and the angle α calculated after acquiring the target, is passed to Triskar function, to calculate the motor contributions on-board, to reach the target.

The inverse kinematics model is simple. It was considered that the representative coordinates of the robot were located in its center. Each wheel is placed in such orientation that its axis of rotation points towards the center of the robot and there is an angle of 120°between the wheels. The velocity vector generated by each wheel is represented on Figure 4.1 by an arrow and their direction relative to the Y coordinate (or robot front direction) are 30°, 150°and 270°respectively.

For this type of configuration, the total platform displacement is achieved by summing up all the three vectors contributions, given by:

$$\overrightarrow{F_T} = \overrightarrow{F_1} + \overrightarrow{F_2} + \overrightarrow{F_3}$$

First of all, some definitions need to be considered. Figure 4.1 represents the diagram of the mobile robot platform with the three wheels. It was assumed that the front of the robot is in positive Y direction, and the positive side to counter clockwise. The three wheels coupled to the motors are mounted at angle position -60, +60 and +180 degrees, respectively. It is

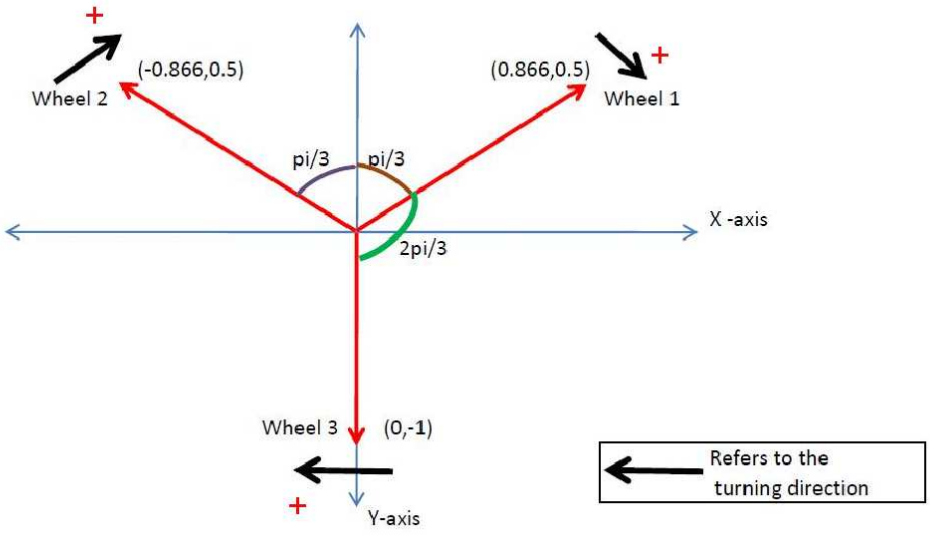

Figure 4.1: The wheel position and robot orientation

important to remember that the wheel driving direction is perpendicular to the motor axis (therefore 90 degrees more). The line of movement for each wheel (when driven by the motor and ignoring sliding forces) is represented in Figure 4.1 by solid, black arrows. The arrow indicates positive direction contribution. The total platform displacement is the sum of three vector components (one per motor) and is represented as a vector, applied to the platform body center.

In order to find out the three independent motor contributions, the composition of the vectors represented by red arrows is projected on axes representing the line of movement of each wheel.

The calculation of the independent motor contributions is made using the following formulas:

$$\begin{bmatrix} V_{t1} \\ V_{t2} \\ V_{t3} \end{bmatrix} = M_F * V$$

where $\mathrm{M_F} = \begin{bmatrix} -cosA & sinA & -1 \\ cosA & sinA & -1 \\ 0 & -1 & -1 \end{bmatrix}$, $\mathrm{V} = \begin{bmatrix} V_F \\ V_L \\ omega * R_{\mathrm{robot}} \end{bmatrix}$

A is the angle of the front wheels, which is 30°for our robot. Indeed the values (0.866, 0.5 ; -0.866, 0.5 ; 0, -1) in the Figure 4.1 coming from the projection of cosine and sine of the motor vectors in the X-Y plane, which are later used statically in the microcontroller for the calculation of the motor contributions. V_F represents the frontal speed, V_L lateral speed and $omega * R_{\mathrm{robot}}$ represents the angular velocity of the body.

The vector of tangential wheel speed is represented as $\mathrm{V_{t1}}$,$\mathrm{V_{t2}}$,$\mathrm{V_{t3}}$ and vector of "sliding" velocity of the wheels due to rollers is represented as $\mathrm{V_{n1}}$,$\mathrm{V_{n2}}$,$\mathrm{V_{n3}}$. The vectors can have a positive or negative direction which represents the direction in which the motor has to move (forward or backwards respectively). The desired speeds set as frontal speed (V_F) and lateral speed (V_L) are projected through the motor axes in order to find motor contributions.

The angular velocity of the wheels is found by dividing the V_t of the desired wheel to the radius of the wheel. It can be formulated as follows:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} V_{\mathrm{t1}} \\ V_{\mathrm{t2}} \\ V_{\mathrm{t3}} \end{bmatrix} / R_{\mathrm{wheel}}$$

4.2 Matlab Script

In order to test the behavior we implemented a software simulator in Matlab, that is calculating the motor contributions in order to go a specified position in the world. It is possible to obtain three different movement model using the inverse kinematics model. These are:

- Linear Movement
- Rotation
- Mixed Angular and Linear Movement

Linear Movement

This is case where the angular speed of robot should be zero, and there should be a linear movement in the displacement, which is the composition of V_F and V_L vectors.

The software simulator that we built calculates the motor contribution. The user inputs three variables (frontal speed, lateral speed and angular speed) and the program outputs each motor contribution. Figure 4.2 shows the result of the simulator for the linear movement forward.

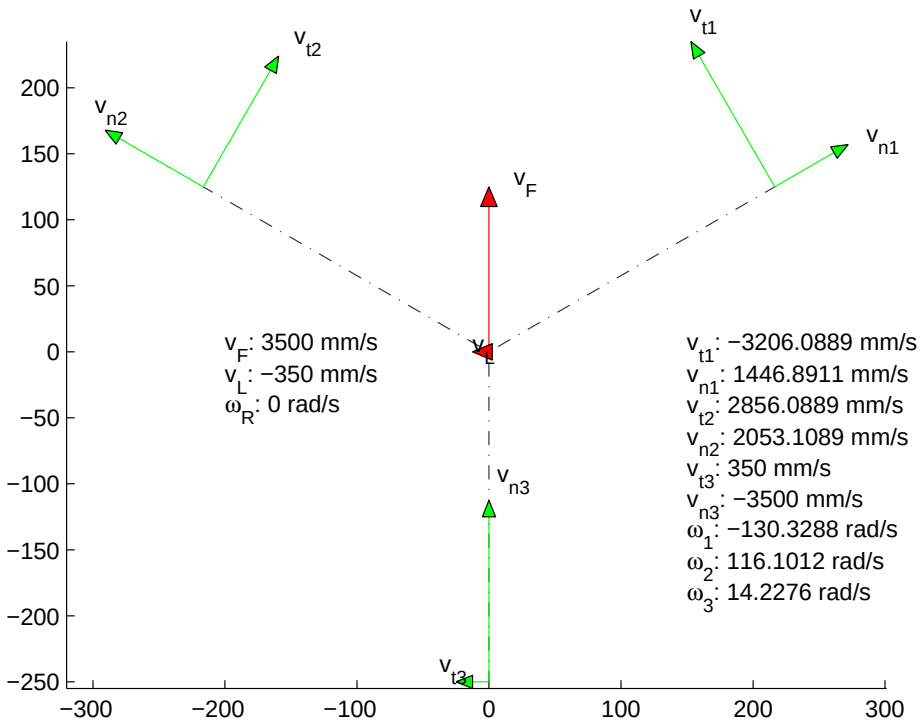

Figure 4.2: A linear movement going slightly on the left on a line calculated by simulator

In Figure 4.2 V_F represents the frontal speed is set to 3500 mm/s, V_L lateral speed to 350 mm/s and W_R represents the angular velocity,which is always 0 rad/s for the linear movement. The resulting motion is a linear movement towards the direction which is the composition of V_F and V_L. The motor contributions and their angular velocities are also shown in Figure 4.2.

Rotation

The second case is the rotation. For that particular configuration, we only considered the movement caused by rotation without any displacement. In the software simulator, the frontal speed and lateral speed are set to zero, and the angular speed is calculated from the robot's pose α using (4.1). An example outcome of the simulator is shown in Figure 4.3. The robot is moving around its center with an angular speed of -0.099669 rad/s, without any displacement in X-Y plane.

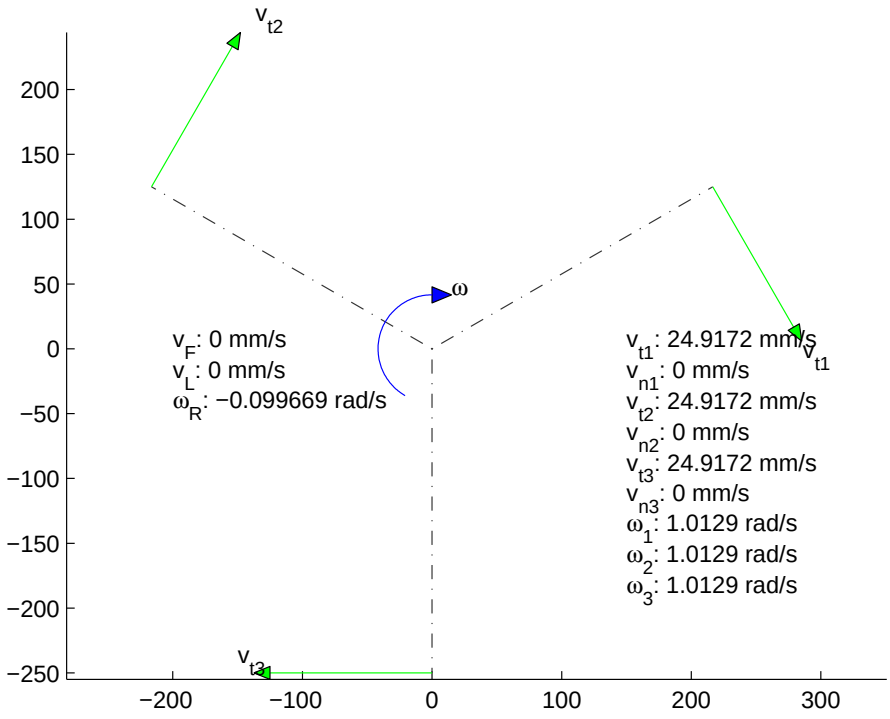

Figure 4.3: The angular movement calculated by simulator

All the theory for the calculation of the angular movement is same as in the linear movement case, only values for some parameters change. The angular velocity of the robot body ω_R is calculated with α in the hybrid approach we mentioned in the beginning of the section.

Mixed Angular and Linear Movement

This is the case where the movement of the robot is composed of both the angular and linear movement. In other words, the robot goes to the specified target while it is also rotating. The calculation made by the software simulator uses all the parameters (frontal speed, lateral speed and angular movement). The calculation of the frontal and lateral speed is found by subtracting the desired position of the robot from the initial position (Figure 4.4).

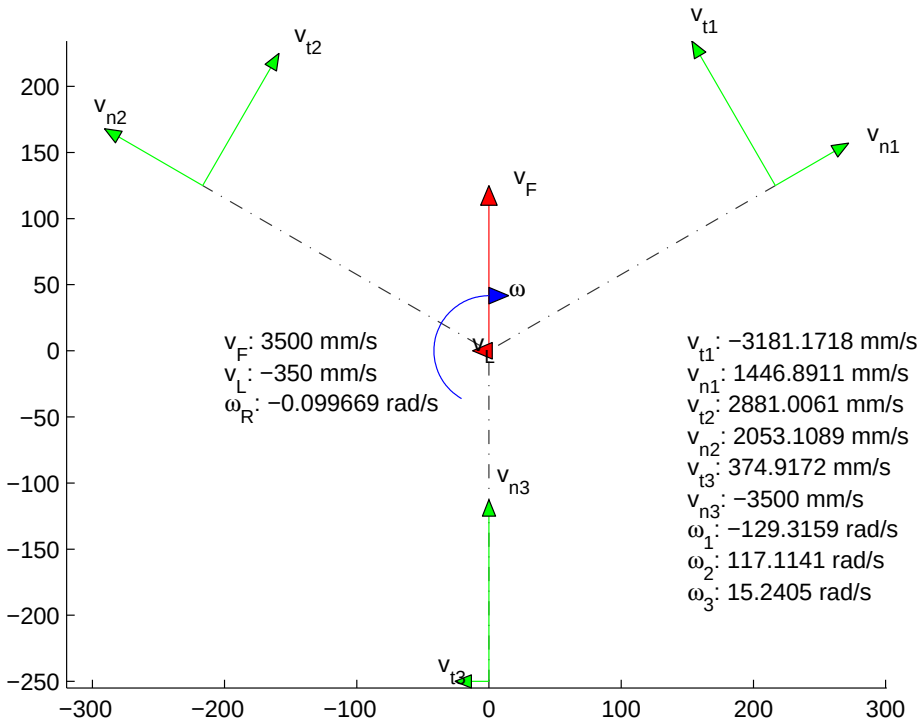

Figure 4.4: The mixed angular and linear movement calculated by simulator

In Figure 4.4, the resulting motion is not only a linear movement towards the direction which is the composition of V_F and V_L, but also an angular movement in clockwise direction with an angular speed of -0.099669 rad/s.

All these three different configurations are used in the implementation of the robot. The whole script is found in Motion Simulator in Appendix

B.3. The most common used one is the mixed angular and linear movement. Almost every path was initially implemented with this movement approach. Others have been used where optimization is needed to cover problems due to the lack of encoders to control the position of the motors.

4.3 PWM Control

Motors are controlled by applying a PWM signal to the H-Bridges. In order to control velocity we did not use any encoders, so we don't have feedback about the motor rotation, or the displacement achieved with the result of the applied PWM. To have an odometry information without using encoders, we decided to review and replicate the work done at AIRLab about using the mice odometry to get feedback from the motors. On the previously work, a set of mice were reconfigured to obtain the position information, and calculations were made from these readings to find the odometry. The working configuration was able get a feedback when the distance between the mouse sensors and lens was about 1mm, and the distance between lens and ground was also 1 mm, but this distance is too small for our robot. In order to increase the distance to 3 cm which is the distance between the ground and the chassis of the robot, we experimented changing the lens of the mice.

The mice work like a camera, and the image is captured by the sensor inside it through a lens placed in front of it. We tested lenses with different focal lengths and different orientations in order to achieve the 3 cm configuration. During the experiments, we obtained different performance on different grounds and different lighting conditions. We tried several illumination alternatives, such as laser, led, and table lamp. We were able to find the correct illumination using the led with a wave length readable from the sensor. And for the lens we used a configuration with a focal length of 10 mm, leaving the distance between the lens and the sensor at 15 mm and lens to ground at 30 mm. Instead of using encoders that each cost around 50 euro, the possible configuration costs around 5 euro, but the work is not finished yet and we decided to include this on the robot later.

Since there is no information on the motor speed and displacement of the robot, the control process mainly depends on vision, and it is not very precise. There are situations where we cannot determine whether the motor is moving. When the robot is stuck in a place and the sensors do not detect

a collision, such as entrance of the wheels to a gap in the floor, since we don't have a feedback on the motors we cannot determine whether the motors are turning.

The motors are controlled by PWM generated from the microcontroller. To drive motors we used a PWM signal and varied the duty cycle to act as a throttle: 100% duty cycle = full speed, 0% duty cycle = coast, 50% duty cycle = half speed, etc. After some testing we realized that the percentage of the duty cycle and the speed are not changing with the same ratio (e.g. 50% of the duty cycle does not correspond the half speed). Also there is a minimum percentage of the duty cycle which the wheels start turning, and a maximum percentage of the duty cycle at which the speed remains as full speed after that. The minimum PWM value to start the motion is % 45 of the duty cycle and the speed set to full speed after exceeding the % 90 of the duty cycle which is the maximum PWM that can be applied. Since there is a non-linear increase between the ratio of rotation of the motors and applied PWM rather than a linear one, the speed changes are not distinguishable. In other words, the range of PWM signal that enables movement is very narrow and the difference between the minimum speed and maximum speed is not very apparent.

The script that is calculating the motor contributions according to the target position is not taking into account the motor's maximum or minimum speed. The script can return a motor contribution with 4000 mm/s or 3 mm/s which is cannot be performed physically by the motors. After some experiments, to limit the script that calculates the motor contributions, we decided to bound the PWM signal value to be applied to the motors. The maximum PWM value defined for a motor is 2.5 meters/second and the minimum PWM value is around 300 millimeters/second. Using both the limits coming the physical characteristic of motor, and the PWM-rotation limitations, we implemented a map that transforms the motor contribution calculated by the script into a PWM signal. To do so, all the speeds between 2500 mm/s and 300 mm/s should be mapped to PWM signals between 45% and 90% of duty cycle. Mapping 300 mm/s to the 45% and 2500 mm/s to 90% did not resulted as expected since reflecting speed changes is not possible.

In order to express most of the speeds, we decided to tune the PWM sent to motors. Main idea is to bound to speeds in a dynamically changeable manner. We introduced, a fuzzy-like behavior that is based on the robot's

distance to the target position. Each distance is divided by a constant, then multiplied with the multiplier defined as FAST, NORMAL and CLOSE. Then another constant which is the minimum value that can initiate the motors movement is added. Hence, the mapping that each value set by the PWM signal generates a motion, is achieved. The multipliers are defined according to the distance of the robot to the target object. For the distances between 1200 mm and greater, the corresponding multiplier is classified as FAST, between 1200 mm and 750 mm the multiplier is NORMAL, and for the distances between 750 mm and 300 mm it is CLOSE. By defining such control mechanism as FAST, NORMAL, CLOSE we are able to create three different speed limits for the motors, which are all mapped from 60% to 85% of the duty cycle where the motion is visible. As an example case, when the robot detects a target at 3 meters, the motor speeds will be set with multiplier FAST until the distance to the target reaches 1.2 meters. When reached to 1.2 meters, the motor speed will be set with multiplier NORMAL until the detected target distance is less than 700 millimeters. And finally, the motors speeds will be set with multiplier CLOSE from 700 mm until the target is reached. This mechanism allows to achieve more precision in the motion, since the mapping guarantees that each motor speed set by PWM will generate a reasonable motion. Moreover, instead of trying to map all the speeds uniquely, indeed we are trying to map only one third of the speeds, from the predefined speed ranges to 60%-85% of the PWM value in the duty cycle. Even though the robot goes to target with an unexpected path, the path is corrected as the robot comes closer since the slow motor speeds are more precise and easy to achieve.

Even though the solution is not working perfectly, the robot is able to go to the target successfully almost every time. The travel time and number of movements in order to reach the target are not optimal, but this result should be regarded as normal since we are not using any encoder to detect the movement in the motors.

The need for an optimization of the PWM signal is also related to the result of the miscalculation of the required torque. We selected motors that do not have enough torque power for our robot. This brought the problem of the arrangement of the PWM signal in order to run the motors. Since the motors have less torque than we expected, almost 50% of the PWM is wasted. First trials to optimize this inefficiency have been made by changing the frequency of the PWM to a higher or a lower frequency, but these trials did not give good results. Later, we found the proper configuration,

mentioned previously, by limiting the minimum and maximum PWM values of the motors and by introducing the fuzzy-like control to introduce different limits for the different distances to target. Even though the lack of the encoders reduced the performance, we were able to produce an acceptable configuration.

Another step is initialization of the motors. The corresponding pins in the microcontroller for the motors are set to control the direction of motors and PWM generation. The PWM generation is made by using the timers available in the microcontroller. Output compare mode of the timers is used in order to create the PWM needed to run the motors. Output compare function is used to control the output waveform and indicates when a period of time has elapsed. When a match is found between the output compare register and the counter, the output compare function:

- Assigns a value to pins
- Sets a flag in the status register
- Generates an interrupt

Using the described output comparison, we created the desired PWM wave to run the motors. At, first the timer clock is set to 8 mHZ. Then output compare register 1 is set for the desired motor speed as some percentage of the full period of the cycle. Output compare register 2 is set to full cycle that will set the pin to HIGH, and update the desired motor speed of the output compare register 1. When counter reaches the defined value, the motor speeds will be updated and the pins will be reset. It will remain LOW until output compare register 2 value (which is the full period) is reached. Using that algorithm we generate the PWM signal to run the motor. For each motor, we decided to use a separate timer, since the timers are available to use and not needed by other applications. It is also possible to use only one timer to generate the PWM's for all the motors using the interrupts and output compare registers, but we decided not to use like this for the simplicity of the algorithm, since this will need a better synchronization, and optimization to run efficiently.

The control of the servo that is changing the camera head pointing direction is also implemented using PWM signals. For the servo PWM, we don't have the problem of control that we faced with the motors. So, no tunings

are implemented for this case, and the PWM creation is implemented simply as follows. The PWM which controls the servo position is implemented with the first timer at with the motor. The period of PWM in first timer, to control the motor, is divided by a constant in order to achieve a smaller PWM period, which is around 2ms for the servo. The rest of the idea is the same. The PWM is generated with the help of the output compare mode. The servo position is initialized at the same time with the motors, then the position is controlled calling the Position_Camera(); function that can take SERVO_TOP, SERVO_MID, SERVO_BOTTOM as the different positions for the camera servo. Indeed this camera positions are used during the vision inspection to reach the target and calculate the motor contribution according the distance of the target object. 3 unique camera head position are implemented using the visual feedback from the ST software, to find the best configuration in terms of depth of field and field of view. If the target is more far than 750 mm the camera position is set to SERVO_TOP, which can see the environment from 750 mm to robot up to 4-5 meters. If the target is closer than 750 mm to robot, the camera head position is lowered to SERVO_MID, in order to provide a vision from robot boundary to 750 mm. The last position is implemented for testing reasons, and also provides a ready configuration to set the camera head in any position.

Chapter 5

Vision

The vision system is focused on the implementation of the camera for object detection and calculating the position of this object according to the robot. To do so, the relation between different coordinate systems, which are image plane coordinate system, robot coordinate system, and real world coordinate system, should be defined. In order to go to the target, it is necessary to know where the object is placed. Using the blob search algorithm, we acquire the position of the object in the pixel coordinates. We need a mapping between those three coordinates to command the movement. There are several approaches to determine the object position according to the robot, as mentioned in Chapter 2. Among those, we will use the one for the calibrated cameras.

The rest of Chapter describes camera calibration, the script that calculates the transformation, color definition, and the script to define the color.

5.1 Camera Calibration

The initial step for the algorithm is to find the camera calibration. The Camera Calibration Toolbox [2] is used to calculate the camera calibration matrix that is necessary to find the projection between the world and image points represented by homogeneous vectors. Normally, the object is placed on the real world coordinate systems, and its position can be measured from the origin. In order to find the distance of the camera plane to the object, we need a transformation that maps the 3D points (for the object that is assumed to place in ground, the point is a 2D point in world coordinates, since height is 0 all the times) of the objects in the 3D world coordinates to

the 2D points in the image plane coordinates. To find the transformation matrix, we use the camera calibration matrix, calculated from Camera Calibration Toolbox [2]. In general, the Camera Calibration Toolbox works as follows. The sample pictures of the checker board it taken by the camera in order to calculate the camera matrix (Figure 5.1). By taking the pictures of the checker board makes it possible to use the information of its dimensions in order to calibrate the camera, by that the unknown parameters of the camera can be calculated such as focal length, principal point, skew, radial distortion, etc.

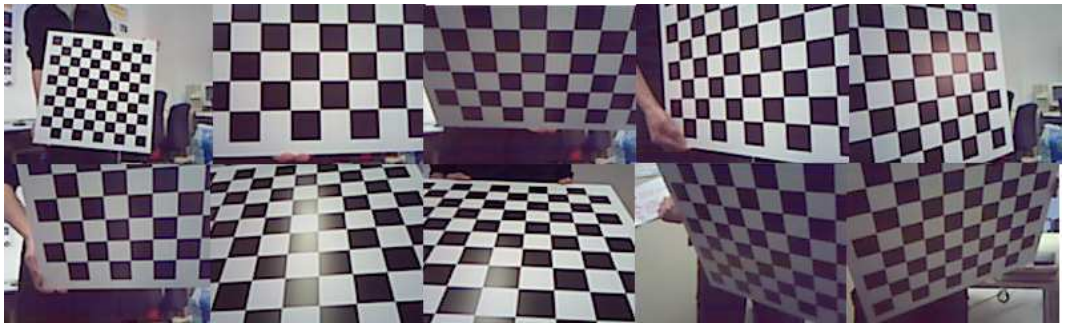

Figure 5.1: The images used in the camera calibration

In order to calculate the homography matrix, which makes the transformation of the points from camera coordinate to robot coordinate, we wrote a script that makes the calculation for us. Having the camera calibration matrix from the toolbox, the homography matrix (H) is calculated by the script.

The whole camera calibration and the projection between the world points and camera as explained in details as follows. The matrix K is called the camera calibration matrix.

$$K = \begin{bmatrix} f_x & & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix}$$

where f_x and f_y represent the focal length of the camera in terms of pixel dimensions in the x and y direction respectively. Similarly, p_x and p_y are

the principal points in terms of pixel dimension.

$$x = K[I|0]X_{\text{cam}}$$

where X_{cam} as $(X, Y, Z, 1)^{\text{t}}$ to emphasize that the camera is assumed to be located at the origin of a Euclidean coordinate system with the principal axis of the camera point straight down the z-axis, and the point X_{cam} is expressed in the camera coordinate system.

In general, points in the space will be expressed in terms of a different Euclidean coordinate frame, known as the world coordinate frame. The two frames are related via a rotation and a translation. The relation between the two frames can be represented as follows:

$$x = KR[I| - C]X$$

where X is now in a world coordinate frame. This is the general mapping given by a pinhole camera (In Figure 5.2). A general pinhole camera, $P = KR[I| - C]$, has a 9 degrees of freedom: 3 for K, 3 for R, 3 for C. The parameters contained in K are called internal camera parameters. The parameters of R and C which relate the camera orientation and position to a world coordinate system are called the external parameters. In a compact form, the camera matrix is

$$P = K[R|t]$$

where $t = -RC$.

The camera matrix, which consists of internal and external camera parameters, is calculated by the camera calibration toolbox automatically. Having the internal and external camera parameters from the toolbox, we compute the homography H. Homography is an invertible transformation from the real projective plane to the projective plane.

We developed two different approaches to determine the position of a target according to the robot coordinates. In the first case, the target object is a ball, and we are using the information coming from the diameter of the ball. In this case, the transformation should be a 3D to 2D, since the ball can be represented in world coordinates system with the X,Y,Z and in the camera coordinates with two values X,Y. In the second case, we assume that the target object is on the ground and we are using the information coming from the intersection point of the object with the ground. This

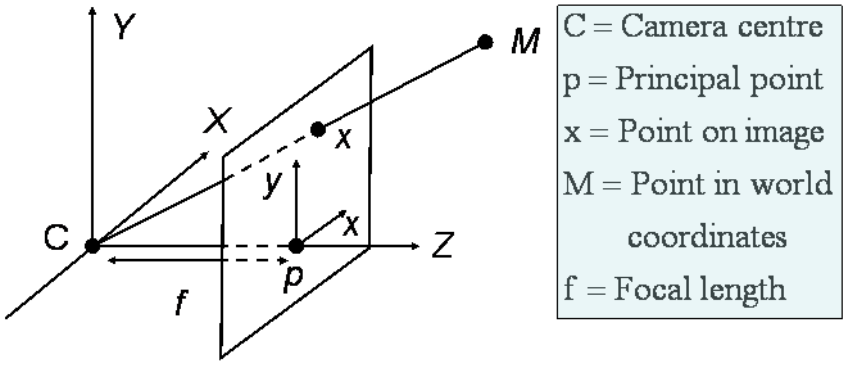

Figure 5.2: Perspective projection in a pinhole camera

means the object can be represented by a X,Y,Z position in the world coordinate system, with Z=0. This reduces the transformation into a 2D-2D transformation, and needs a new H matrix.

We need to calculate the transformation matrix between the robot and camera (T_C^R) coordinates. The transformation matrix between world frame and camera (T_W^C) coordinates is known from the external camera parameters. The world frame to robot frame transformation matrix T_W^R is calculated by the target object position and robot center (shown in Figure 5.3). T_C^R is derived as follows:

$$T_C^R = T_W^R * T_C^W$$

For the ball localization case, the calculation is made for the pixel diameter D_{px} of the ball at known distance l and the real diameter D_{ball}. For the camera, we introduced a parameter f^*, to indicate the dimension of a unit pixel, which is a statical parameter of the camera. Using this information, the distance of the ball l_{new} can be calculated by counting the pixels of the diameter of the ball D_{px} and by multiplying this by f^* .

$$f^* = \frac{l}{D_{\text{ball}}} * D_{\text{px}}$$

$$l_{\text{new}} = \frac{f^*}{D_{\text{px}}} * D_{\text{ball}}$$

The next step is converting the direction from camera to the ball $\overrightarrow{d^c}_{\text{ball}}$

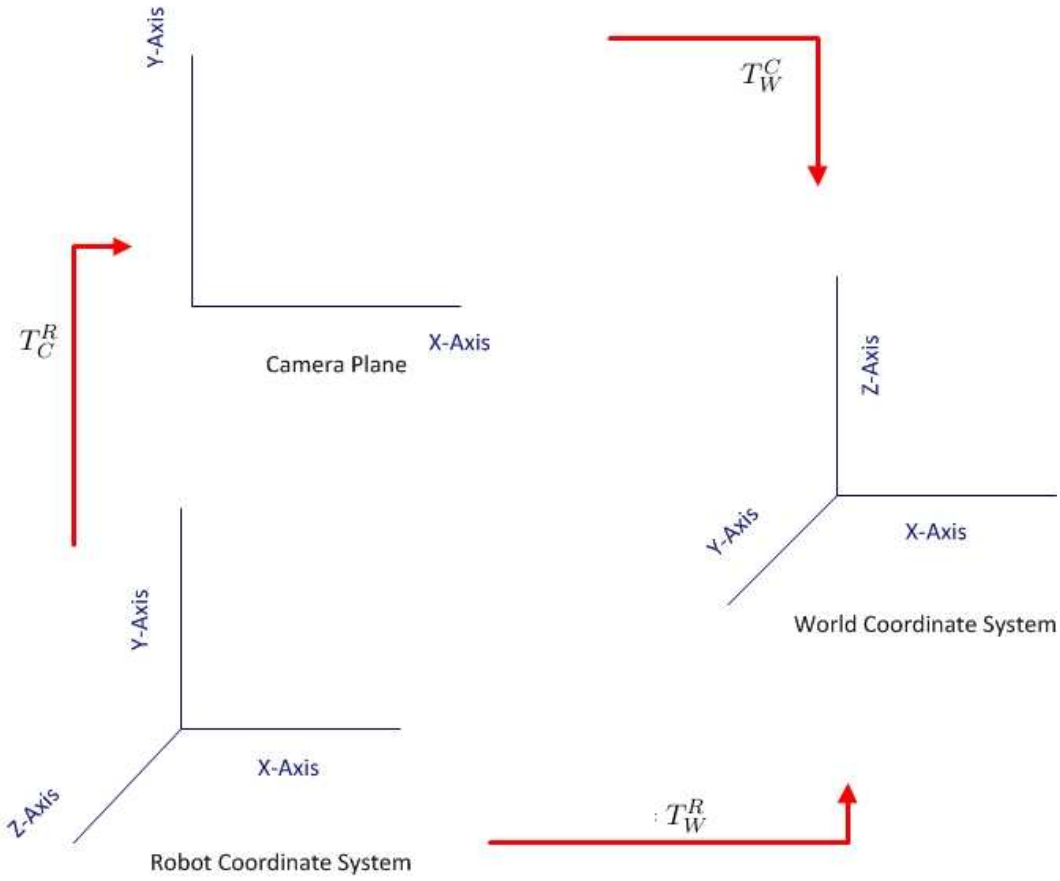

Figure 5.3: Camera robot world space

into the direction vector, and its normalized vector $\overrightarrow{d^c{}_{\text{norm}}}$ in order to find the direction from camera plane to the ball.

$$\overrightarrow{d^c{}_{\text{ball}}} = K^{-1} * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$\overrightarrow{d^c{}_{\text{norm}}} = \frac{\overrightarrow{d^c{}_{\text{ball}}}}{|\overrightarrow{d^c{}_{\text{ball}}}|}$$

From the direction $\overrightarrow{d^c{}_{\text{norm}}}$ and the line l_{new}, the point $P^c{}_{\text{ball}}$ that is 3D homogeneous coordinates of the points with respect to camera.

$$P^c{}_{\text{ball}} = \begin{bmatrix} l_{\text{new}} * \overrightarrow{d^c{}_{\text{norm}}} \\ 1 \end{bmatrix}$$

To calculate ${P^R}_{\text{ball}}$ which is 3D homogeneous coordinates of the point in world, the transformation matrix camera to robot (T_C^R) is used.

$${P^R}_{\text{ball}} = T_C^R * {P^C}_{\text{ball}}$$

The second configuration is for the object on the ground. This case is simpler since it requires a 2 D to 2 D transformation. The transformation between camera and robot is used again and the calculation is the same as the previous configuration. The homography is defined for this case as:

$$H = K * T_R^C * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The third row of the matrix is all zeros since the we assume that the object is at ground with Z=0, hence this reduces the mapping from 3D to 2D. To calculate the object position P_{object}, we will use the inverse homography on the image point P_{image}.

$$P_{\text{object}} = H^{-1} * P_{\text{image}}.$$

P_{object} is then normalized. The resulting point is the distance of the robot to the target object. This information is later used to calculate the motor contributions.

Among the mentioned approaches, we started with the first configuration (for the ball). The idea was to detect a ball, no matter whether it is on the ground or up in the air. We started by testing with a red ball, since the color red, blue, green were initially defined for the camera. The results were not sufficient to follow the ball for greater distances because of several effects. One of them is related with the camera resolution. Normally the camera can work with 160x120 resolution for the blob search. This does not cause a problem if the ball is close to the camera, but as the ball goes further from the camera, the ball represented in the image is getting smaller and smaller. Due to the low-cost optics, at the worst case the ball becomes 1 pixel in the picture. Using only this pixel, it is impossible to detect the ball. Even though, we can recognize the ball for some cases, in most of the cases we cannot. Due to the low-cost optics, it is possible to have several other 1 pixel size red blobs that are not our ball. In order to detect the ball, inside the blob search algorithm we need to eliminate the blobs that are not the ball we are searching.

Later, in order to solve this problem, we came up increasing the size of the ball. Again, this trial did not resulted as we expected, since when the ball gets further from the robot, the diameter information is not reliable. This results to calculate a wrong value of the ball position. In this trials, we realized also the effects of the white balance and exposure compensation for the color information. The camera tries to automatically adjust the color information as the ball gets further from the light source. Since the blob elimination is dependent on both the shape of the ball and the color of the blob, the information changed by the camera results to definition of wrong blob information. To clarify with an example, the ball shape is checked by the circularity information calculated by the camera, but the whole ball is not composed of the same color, since the light is affecting the color, so the center can be more bright or the corner more dark etc. The camera with the automatic setting of the white balance and exposure compensation tends to change the color in order to have a better picture, but by doing so the color temperature is changed. So the blob search eliminates the non-color points as well as the non-shape objects and detects the ball in another shape, such as an open rectangle. This whole change also results to a change of the diameter of the ball and the distance information is not calculated correctly.

The diameter of the ball is a problematic issue due to the low-cost optics. Instead, we decided to change our ball into a cylindric shape and assume that the object will always be on the ground. This reduces the mapping to a 2 D to 2 D mapping problem, since Z (the height) will be always zero. Instead of calculating the diameter of the ball, we determine the distance between the end of the image and the intersection point of the object with the ground. This distance is easier to measure, independently from the shape of the object, and will always change as the object's position changes in the world. The mentioned mapping also makes easier the blob search, since it is not very important to detect the object clearly as a cylindric. If we can recognize some part of the object we can ensure that the object is more or less close to the calculated point. This also gave us the flexibility for the color information, since we do not have the obligation to detect all the object, we can define a more generic color code that is working for most of the cases.

We used the second configuration in both camera positions. Even the internal camera parameters are always same in our camera, the external parameters are effected with a translation or rotation. Indeed, when the camera head position is changed, it results with the usage of a H matrix for each transformation.

The whole algorithm and script is found in Object's Position section in Appendix B.2

5.2 Color Definition

The camera we are currently using has a UXGA resolution CMOS and it is set to 160x120 resolution for the blob search algorithm. One major problem is the blob search algorithm. The algorithm used inside is unknown and this forced us to define color information very precisely, since we cannot eliminate the non-object blobs easily. We can eliminate the problem only by using the area, perimeter and the circularity information which, in case the target is a ball, is not enough for most cases to track the blob. So this makes the color definition important. The color is defined in a RGB Cube, using RGB444 format, and it is not very clear to understand the correctness of the defined color from the color codes. In order to ease this process, the color selection is made by the ST software by selecting the desired color from the image, or directly programming the rules if the color values are known. Since the resolution is very low, the color consists of shadows, or existence of blob not belonging to the searched object made us to improve the color selection. The current ST software is sufficient to visualize the selected color and test the blob search, but color definition is difficult since, in order to define the color, the pixels should be selected one by one, is not possible to select an area. Moreover, the rules (which is the additions and subtractions of the color codes to RGB Cube) found by the ST software are not in the compact form (the same rule for a color can be added or subtracted more than one). To improve the color selection, we decided to use an offline calculator, that calculates the histogram for the selected object's colors.

As a first step, we started by disabling automatic white balance and automatic exposure control. Before getting into the white balance, the concept of color temperature needs to be introduced. Color temperature is just a way of quantifying the color of light. It is measured in degrees Kelvin (K). Normal daylight has a color temperature of around 6,500K. Warmer light has a lower color temperature. The warm light that occurs late in the afternoon might have a color temperature of around 4,000K. Cooler light has a higher color temperature. The bluish light that sometimes occurs in twilight periods of the day might have a color temperature of about 7,500K. So, our concept of warm and cool light is tied directly to the color temperature. The

warmer (yellow) the light, the lower the color temperature; the cooler the light (blue), the higher the color temperature.

Color Temperature	Light Source
1000-2000 K	Candle light
2500-3500 K	Tungsten Bulb (household variety)
3000-4000 K	Sunrise/Sunset (clear sky)
4000-5000 K	Fluorescent Lamps
5000-5500 K	Electronic Flash
5000-6500 K	Daylight with Clear Sky (sun overhead)
6500-8000 K	Moderately Overcast Sky
9000-10000 K	Shade or Heavily Overcast Sky

Color temperature is also important for cameras. The camera manufacturers knew that the color of the light would affect the colors delivered by the camera. Therefore, they decided to deal with the problem by designing the cameras to automatically measure the light temperature and to make adjustments as the light changes color. That is why we can shoot in the relatively neutral light with camera in the afternoon and then shoot the next day in the cool light of early morning and still, probably, get reasonable colors in both situations even though the color of the light was different. Cameras correct for the change in light temperature, using white balance.

With auto white balance, the camera attempts to determine the color temperature of the light, and automatically adjusts for that color temperature. Auto white balance works reasonably well under the following conditions:

- The application does not require absolute maximum color accuracy
- There is not a preponderance of one color in the scene being photographed
- Adjustments for the color temperature of the light

As mentioned in the previous paragraph, in auto white balance mode, the camera does its best to determine the color of the light and make appropriate adjustments. However, the methodology that is used to do this requires that certain assumptions be made. These assumptions do not always match perfectly the scene being photographed. As a consequence, the auto white balance option does not always yield perfect results. Accordingly, problems

may occur using auto white balance when the conditions listed above are violated. Therefore, auto white balance may not be a good choice if:

- Absolute color accuracy is required
- There is a lot of one color in the scene: The preponderance of one color can fool the auto white balance function into assuming that the light has a lot of that color in it. This can result in an incorrect white balance and a color cast.
- No adjustments made for the temperature of the light wanted: In certain cases, the color of the light is what makes the photograph. A sunset is an example. Without the rich, warm colors of the light, a sunset just isn't a sunset. Auto white balance may attempt to make adjustments to correct for the warm color of the sunset light. This would produce an image with less saturated colors or colors that were different than what has been seen.

The camera we are using supports different settings for the balance. These are:

- OFF - No White balance, all gains will be unity in this mode
- AUTOMATIC - Automatic mode, relative step is computed here
- MANUAL_RGB - User manual mode, gains are applied manually
- DAYLIGHT_PRESET - DAYLIGHT and all the modes below, fixed value of gains are applied here.
- TUNGSTEN_PRESET
- FLUORESCENT_PRESET
- HORIZON_PRESET
- MANUAL_color_TEMP
- FLASHGUN_PRESET

Among the possible options, we set the white balance OFF. Because, in the application we require absolute color accuracy and we don't want the camera to change the temperature of the color since we want to detect the target with a constant color parameter defined using the samples we took.

Another automatic adjustment can be done on exposure. Subjects lighter than middle gray, such as a white china plate, reflect more than 18% of the light falling on them. The exposure system doesn't know that the scene should look bright, so it calculates an exposure that produces a middle gray image that is too dark. Subjects that are darker than middle gray such as black cloth, reflect less than 18% of the light falling on them. The exposure system calculates an exposure that makes the image middle gray and too light.

The contrast or difference in brightness between the subject and the background can fool an exposure system, particularly if the subject occupies a relatively small part of the scene compared to the background. The brightness of the background is so predominant that the automatic exposure system adjusts the exposure to render the overall brightness as a middle gray. If the main subject is lighter than the background, it will be overexposed and too light. If it's darker than the background, it will be underexposed and too dark.

Depending on the arrangement of the lighting, some subjects may be too contrasty with brightly lit highlights and deep shadows. The range of brightness may exceed the range that can be captured by the camera. In these cases adjustments should be made in the lights to balance out the light and to lower the contrast. However, deciding whether the highlight or shadow areas are most important for the final picture, the exposure setting should be made appropriately.

The perfect exposure retains details in both the highlights and shadows. For the auto exposure system, this is as difficult. If there is even a little too much exposure, the image is too light and details are lost in the highlights. If there is too little exposure, the image is too dark and details are lost in the shadows.

When confronted with any subject lighter or darker than middle gray, exposure compensation is used to lighten or darken the photograph that the camera would otherwise produce.

To lighten a picture, the exposure is increased. This is useful for setups where the background is much lighter than the subject, or when photographing very light objects, such as white china on a white tablecloth. To darken an image, the exposure is decreased. This is useful for setups where the

background is much darker than the subject, or when photographing very dark objects, such as black china on a black tablecloth.

From the parameters available in our camera we disabled the automatic exposure by setting DIRECT_MANUAL_MODE, in which the exposure parameters are input directly and not calculated by the camera. We don't want the camera automatically adjust the exposure, since this will cause to lighten or darken the picture dynamically. Instead by setting the exposure directly to a constant value, we ensure that the picture will remain with the same enlightenment, together with the target we are searching for.

After setting the white balance and exposure compensation, we took sample images (Figure 5.4) in order to define selected object's color using the script.

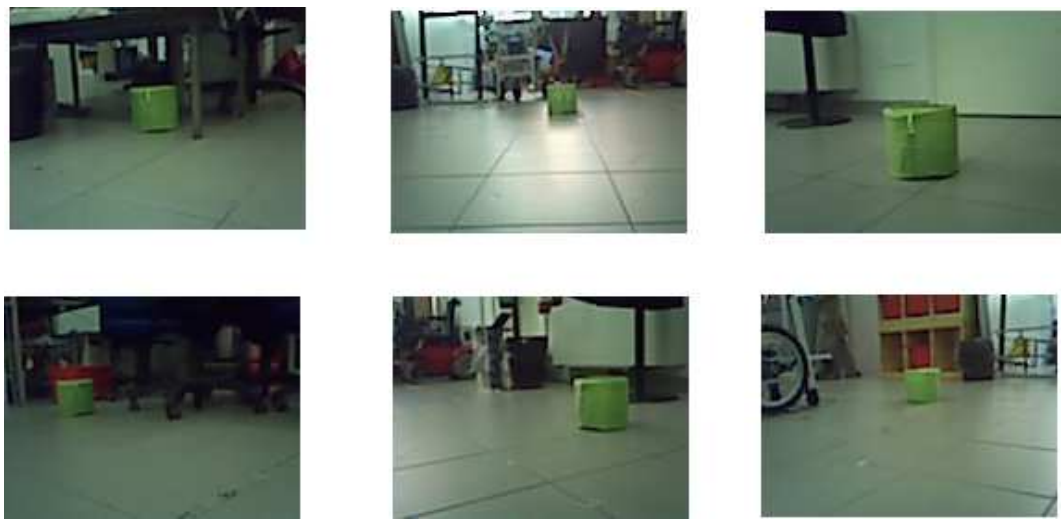

Figure 5.4: Samples are taken from different lighting conditions and different distance to the target object

After acquiring the samples, the next step is creation of the histogram, to find the distribution of the RGB values in the target object's color. The histogram created is shown in Figure 5.5.

An image histogram is a type of histogram that acts as a graphical representation of the color distribution in a digital image. It plots the number of pixels for each color value. By looking at the histogram for a specific image,

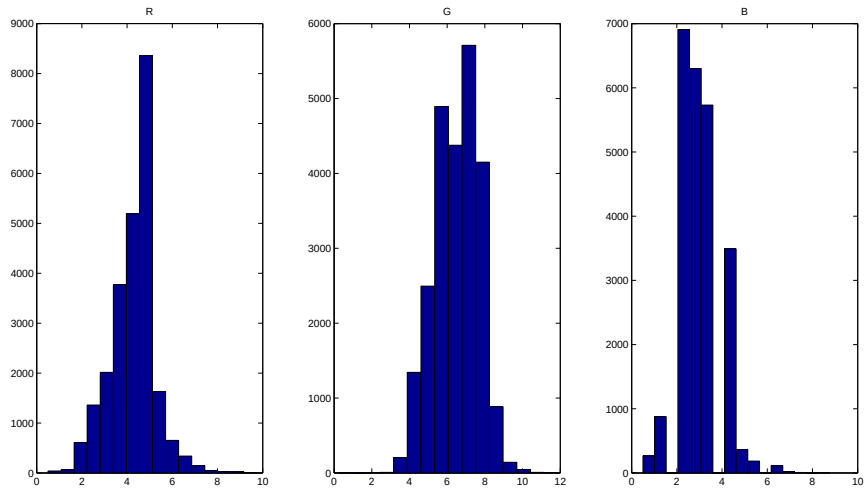

Figure 5.5: The histogram formed from the samples taken for each color channel.

a viewer will be able to judge the entire color distribution at a glance. For this we wrote a script in Matlab. We took as many samples as we could from the environment where the object is placed in different lighting conditions.

From the sample images captured, the part with the object is cropped (area with the target color) in order to find the color distribution of the pixels forming the object.

(a) Mask R (b) Mask G (c) Mask B

Figure 5.6: The mask for each channel by setting the upper and lower bounds.

From that histograms we create the masks (Figure 5.6), by finding the upper bounds and lower bounds for each color channel. As a final step we create the total mask, that is returning the target object's color boundaries

in a form that can be converted into the rules. The result of the color selection script, with the original picture is shown in Figure 5.7.

(a) Mask to be Applied (b) Orginal Picture

Figure 5.7: The mask applied to the orginal picture.

The boundaries found for the object are converted into rules, that will add or subtract the color from the RGB Cube. The rules are checked by color selection program, provided with the ST software coming with the camera. The script can be found in Appendix B.1 in Color Histogram Calculator. Using both the color selecting program and an offline Matlab script, we have been able to define the color in a more precise way, and improved the results for object detection. But the results diverged as the lighting conditions change in the environment. For example the laboratory where we have performed tests is exposed to direct sunlight in the morning, while in the afternoon it is not. We introduced different configurations for the same color, to avoid the need for sampling of the previously defined color. By controlling the color before each run manually and recalculating when necessary resulted a stable solution to the problem.

5.3 Tracking

The final working configuration of the object detection is made by solving the problems described previously. We can explain these in three titles. First one is finding the correct color information. We start by disabling the automatic white balance and automatic exposure compensation. This allows us to have the color as it is seen directly from the camera, by not adjusting

any color values. It means, if the object is exposed to light, the parts that are exposed to light are not considered as white but a tone of target color, similarly with the previous configuration the dark pixel such as the parts in the shadow are considered as black, but with the new configuration they are also considered as a tone of the target color.

Second step is calculation of the color information in a better way. Previously, we calculated the information by ST color picker software. With this software we select the color region we want to assume as the blob color. The problem of this approach is that the blob may not consists of a single rule (RGB color code). Some parts of the object are darker, some parts are lighter. Also with the distance, the color that is composing the object is changing. Additionally, the colors that are not considered as red, blue, green, represented again as rules (RGB color codes) and it is not possible to understand by just looking at the color code whether it is the correct color coding or not. These makes it very difficult to determine the color with the ST color picker software. Instead, we decided to find the color histogram that is the distribution of the selected color properly in the RGB channels separately. To do so, we took several samples from the camera, in different distances, in different illuminations to find a proper average. The calculated results are later tested with the ST color picker software, since it offers a good interface to see whether the color is selected correct or not, in a visual way.

The third step is related to the estimation of the objects distance to the robot. As mentioned before, the diameter of the ball is a problematic issue due to the low-cost optics, hence we decided to use an object with cylindric shape, and the second configuration for the transformation. This distance is easier to measure, independently from the shape of the object, and will always change as the object's position changes in the world.

During the implementation to the microcontroller to increase the performance, we tried to avoid matrix and floating point operations. The matrices that can be defined statically are defined statically. To avoid floating point operations, the floating points are converted into integer, then multiplied by a constant (most of the times 1000). The operations are made in integer, and the result is returned as floating point by dividing with the constant. We ensure that the microcontroller will not spend too much time in matrix and floating point operations, using this approach.

Chapter 6

Game

The robot will be used as a testbed to develop robogames. As a testbed a simple game is implemented, using almost all the components available on the robot. The game is used also to verify the correctness of the solutions implemented. We created a simple game to test the robot. Basically, the robot has to go to several targets in a sequence, by avoiding obstacles, until the final target is acquired.

The game flow can be expressed in a better way with an algorithmic approach. The class diagram is shown in Appendix A.1 and the flow diagram in Appendix A.2. The microcontroller starts by controlling the game end status. The game end status is composed of conditions. First, it checks whether the target is acquired. If it is acquired then it also checks whether this is the last target and ends the game by staying at the current position. If it is not the last target, the algorithm steps from the game status check phase and continues the search. Before performing any other operation the collision status is controlled. If a collision is detected, a proper command is sent to motors to get rid of the obstacle.

The next step is blob search, and blob tracing. The camera searches for the target at its vision side. If no blob is found, the robot performs a turn around its center of rotation until a blob is found or a collision detected. Normally, the collision should not be controlled for the turn around the center of rotation but, as we discussed previously, we cannot guarantee the correctness of the movement since we lack encoders for the motors. When a blob is found, the first step is checking the color information of blob, since the difference between targets is made by the color information. If the cor-

rect target is found, we calculate the distance of the object from the robot. According to the distance found, the motor contributions are calculated and the robot starts going to the target.

We introduced two different camera head positions. At the start the camera is always set at the normal position (which can see the environment from 750 mm from the robot up to 4-5 meters). According to the result of the distance calculated at the previous step, the camera head position maybe be lowered in order to detect and go to the target in the next step. The target is set as acquired when the distance between the robot and the target is below a certain threshold. The whole algorithm we can be seen in Appendix A A.2 in a clear way.

The software that is running the robot is working on-board, on the microcontroller. The software is divided into sub-modules in order to ease the development of the process. In this section, we will give the details of the software, introducing the sub-modules, supporting with the diagrams. The software is composed of low-level and high-level modules. The low-level modules are the assembly code that is coming with the ST tool-chain, which already defines the registers, memory mappings, all the communications with the chips. We did not concentrate on the assembly code; all the written software implements the high level part.

The development mainly focused on the generation of PWM using timers, initialization of components, algorithms and utilities. Like all the microcontroller programs the core runs in the main loop. Before entering the main loop we initialize all the components first. Initialization is made step by step as follows:

- Configure the system clocks
- Configure the GPIO ports
- Configure x24 Camera
- Configure the USB
- Initialize camera parameters
- Initialize motors
- Set initial camera position

- Initialize sensors
- Initialize the blob search and color parameters

In order to initialize the camera, the RGB_Cube which contains the color information and BlobSearch are initialized. We also set the automatic white balance to off and exposure control to direct manual mode. The appropriate parameters for the image format, frame rate, sensor mode and clock are set for the blob search at this point.

The second step is initialization of the motors. The corresponding pins in the microcontroller for the motors are set to control the direction of motors and PWM generation. The PWM generation is made by using the timers were available in the microcontroller. Output compare mode of the timers used in order to create the PWM needed to run the motors. Output compare function is used to control the output waveform and indicate when a period of time has elapsed. When a match is found between the output compare register and the counter, the output compare function:

- Assigns a value to pins
- Sets a flag in the status register
- Generates an interrupt

Using the described output comparison, we created the desired PWM wave to run the motors and camera servo.

The sensors, which are the bumpers for the current state of the robot, are initialized by setting the corresponding pins in the microcontroller.

As a final step in the initialization, the blob search and color parameters are set. The blob search parameters, which are grid size, top, bottom, left and right borders of the image are defined. Similarly the blob search geometry options are defined, which are the color id, minimum-maximum area of the blob, minimum-maximum circularity of the blob and minimum-maximum perimeter of the blob. For each color, the blob search geometry options should be defined separately with the proper coding.

Lastly, the color information should be defined for each color we are searching. The color information is calculated either by the ST color picker

software or from the Matlab script taking samples from the environment in order to calculate the histogram to find the correct color coding. The color values found from the Matlab script should be tested with the ST color picker software in order to check the correctness of the results.

After the initialization phase completed, the main control loop starts. Before entering the main control loop, the parameters, such as search_direction, game_end, color_done, blob_found etc., that are going to be used locally inside the main loop are set to initial values. The main loop starts by reading the values of the sensor data. Later, the 'color_done' status, which is controlling the target acquired condition. Initially, it is set as 'FALSE', since the camera did not acquire any images. Before capturing any image, the 'control_hit' is checked to detect and clear the hit. Until no collision is detected, the capturing of the images will not start. After the capturing is complete, the blobs found with the correct color, if any, are sorted according to the their area, and the one with the maximum area is set as 'myBlob'. If no blob found in this step, 'myBlob' is set to -1 to indicate no blob is found within constraints. The sign of the 'myBlob' is controlled to check whether a blob is found or not. If no blob is found, 'blob_found' or 'counter' status is checked. This part is implemented so that to decrease the number of turns, for each time a blob is not found. Having the 'counter' less then 3, we increment the counter, and do not perform any turns. In the case the 'counter' is greater than 3 or 'blob_found=FALSE', we perform a turn around the center, and set the 'blob_found=FALSE'. That mechanism enabled us to detect a target, that was not detected even if it is on side. Even if the target is on side and it is not detected for 3 times, we perform the turn. To process blob, in 'myBlob' parameter, the first step is controlling the camera position. Depending to the camera position, the position of the target in real world is calculated. The motor contributions are calculated using 'Triskar' function. According to the calculated position of the target the following cases is executed. If the camera is set to SERVO_TOP and the distance of the robot to the target is:

- Greater than 1200 mm.
  The motor speeds from Triskar are set with multiplier FAST
- Between 1200 mm and 700 mm.
  The motor speeds from Triskar are set with multiplier NORMAL
- Less than 700 mm.
  The camera head position is changed to SERVO_MID

If the camera is set to SERVO_MID, the position of the target is calculated with 'CalculatePositionGround' function. The motor contributions are calculated using 'Triskar' function, and the motor speeds from 'Triskar' are set with multiplier CLOSE. If the distance of the robot to target is less than 300 mm, the target is marked as acquired by setting 'color_done = TRUE'.

The detection of target acquired and game-over is done by controlling 'color_done' on every loop. When the 'color_done' is set as 'TRUE', the camera head position is set to SERVO_TOP, 'color_done' and 'blob_found' are flagged as 'FALSE', and color is incremented by 1. Since the current configuration is working with 2 colors, the case where 'color $>$ 2' is controlled for the game-over case. For that case, we introduced a variable as 'reply_counter', which sets the number of replies/rounds to end the game. In the code, this variable is set as 5, that made the game to find and go to the targets 5 times, in the same color order defined. The work flow in the main loop is shown in Appendix A.2.

The class diagram that is showing the relation between the modules is reported in Appendix A A.1. We increased the modularity of the program by separating the code to improve reuse; breaking a large system into modules, makes the system easier to understand. By understanding the behaviors contained within a module, and the dependencies that exist between modules, it's easier to identify and assess the ramification of change. We used the naming convention _Init for initialization classes, functions. The algorithm are implemented inside the Motion and Vision classes. The motion algorithm is implemented by porting the software simulator previously written to the microcontroller. The vision algorithms are also implemented in a similar manner. The camera calibration calculating the position of the object was previously implemented in a Matlab script. Later, the script has been ported to the microcontroller.

The rest of the implementation not mentioned above, focuses on pin assignments (in Appendix D.3), definitions of the constants, creation of the header files.

Chapter 7

Conclusions and Future Work

After the tests performed, proper modifications have been implemented to solve the detected problems discussed previously. At last, we have been able to obtain a working robot, satisfying most of the constraints defined at the beginning. The robot dimensions are more or less same with the designed dimensions. Even though, we don't have a proper method to measure the robot's maximum speed, the maximum speed achieved by the robot is fast enough as expected. The obstacle avoiding is implemented with bumpers, and supported with foams and springs. The camera head is placed on a servo, can be moved up and down. The fully charged batteries provide enough power to move the robot for at least 2 hours. It is difficult to estimate the total price of the robot. The camera board that is used, is not available in the market, and the total cost of the components in camera board might be misleading to determine the cost of the camera board. Moreover, for the other components, the prices show differences as the order quantity changes. In conclusion, the final price might match the cost constraint 250 euro.

The robot is fully working autonomously, playing the designed game without any interruption. The game is finding the predefined colored objects and going to them in the defined order. We tested the game with two objects colored in orange and green. More colors and objects can be introduced easily following the steps indicated in Appendix C. The color detection must be made with the color calculator script and must be tested with the ST color picker software. In order to have stable performance, it

is advised to check the colors before each run, since the illumination of the environment can change the color interpretation. We defined two different configurations for each color, one for the sunny day, the other for the cloudy day, both are working at our testing environment.

In order to solve the problems related to vision, color selection script, a custom white balance and an auto exposure is used to improve the behavior of the system affected by low cost vision. The problems arisen from the motor control is solved by limiting the minimum and maximum PWM values of the motors and by introducing the fuzzy-like control to introduce different limits for the different distances to target. The use of these tunings and optimizations enabled us to select low cost components, which are performing enough well to develop a robot framework that will be used as a basis to implementing different robogames.

As for the future developments, we plan two additions. The first one it is the development of a dock station and implementation of the autonomous charging behavior. The circuit to measure the battery level is already operational. The software to check the battery level and then go to the dock station must be implemented.

The second addition is the mice boards that we are going to use as odometers. Experiments have been made to modify the previous work done at AIRLab, using the mice as an odometry device. We have not been able to design the working prototype, and the control circuit that calculates the position information from the mice yet. Previously, the position information was calculated with a PIC and a PC. The new design will work on the ARM microprocessor, and the interface between the camera board and mice should be designed and implemented.

The third addition is the self adaptation of the robot to battery charge level. Using the output of battery monitoring circuit, the battery level can be measured. These measurements are going to used in the implementation to detect the battery level and change the speed multipliers to keep the robot in the same speed, even if the battery level is going low. Also, the robot is going to adapt itself to make the decision of going to the docking station for reaching, when a low battery status detected.

Implementation of the future developments will result to have a better control in motion mechanism and introduce more autonomous robot behav-

ior, which will be useful in the research line to develop more interesting robogames.

Bibliography

[1] Battle bots-http://www.battlebots.com/.

[2] Camera calibration toolbox for matlab - http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[3] First robotics competition-www.usfirst.org/uploadedfiles/who/impact/brandeis_studies/05ffl_underserved_summary.pdf.

[4] Item-http://www.item24.com/en.

[5] Lego group - http://mindstorms.lego.com/en-us/default.aspx.

[6] Lifelong kindergarten-http://llk.media.mit.edu/projects.php.

[7] Logo foundation-http://el.media.mit.edu/logo-foundation/.

[8] The mars autonomy project-http://www.frc.ri.cmu.edu/projects/mars/.

[9] Microsoft actimates interactive barney to interact with "barney & friends" on pbs-http://www.microsoft.com/presspass/press/1997/sept97/ mspbspr.mspx.

[10] Plexiglass physical properties - http://www.rplastics.com/phprofplac.html.

[11] Robowars-http://www.robowars.org.

[12] Working with aluminum - http://www.search-autoparts.com/searchautoparts/article/articledetail.jsp?id=162604.

[13] Intel® play^{TM}me2cam computer video camera - http://www.intel.com/support/intelplay/me2cam/. *Intel Corporation*, 2008.

[14] irobot®roomba®vacuum cleaning robots. *iRobot Corp*, 2008.

[15] A. Eliazar and R. Parr. Learning probabilistic motion models for mobile robots. In *Proceedings of the twenty-first international conference on Machine learning*, 2004.

[16] N. Tomatis A. Tapus and R. Siegwart A. Martinelli. Simultaneous localization and odometry calibration for mobile robot. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2003.

[17] M. Asada, R. D'Andrea, A. Birk, H. Kitano, and Manuela Veloso. Robotics in edutainment. In *Proceedings of the IEEE International Conference on Robotics and Automation 2000 (ICRA '00)*, volume 1, pages 795 – 800, April 2000.

[18] Bobick et al. The kidsroom. *Communications of the ACM, Vol. 43, No. 3, 60-61*, 2000.

[19] A Bonarini. Designing highly interactive, competitive robogames. *AIRLab report*, 2010.

[20] David Cavallo, Arnan Sipitakiat, Anindita Basu, Shaundra Bryant, Larissa Welti-santos, John Maloney, Siyu Chen, Erik Asmussen, Cynthia Solomon, and Edith Ackermann. C and ackermann e: âroballet: exploring learning through expression in the arts through constructing in a technologically immersive environment. In *in Proceedings of International Conference on the Learning Sciences 2004*, 2004.

[21] M Tan and Y Shen D Xu. A new simple visual control method based on cross ratio invariance. *IEEE International Conference on Mechatronics & Automation*, 2005.

[22] K. Deguchi. Optimal motion control for image-based visual servoing by decoupling translation and rotation. *Proc. Int. Conf. Intelligent Robots and Systems*, 1998.

[23] Carl F. Disalvo, Francine Gemperle, Jodi Forlizzi, and Sara Kiesler. All robots are not created equal: The design and perception of humanoid robot heads. In *in Proceedings of the DIS Conference*, pages 321–326. ACM Press, 2002.

[24] Joseph Djugash, Sanjiv Singh, and Benjamin Grocholsky. Modeling mobile robot motion with polar representations. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems*, IROS'09, pages 2096–2101, Piscataway, NJ, USA, 2009. IEEE Press.

[25] Han et al. A new landmark-based visual servoing with stereo camera for door opening. *ICCAS2002*, 2002.

[26] Frizera Vassallo et al. et al. . Visual navigation: combining visual servoing and appearance based methods. *SIRS'98, Int. Syrup. on Intelligent Robotic Systems*, 1998.

[27] Shirai et al. et al. Wiimedia: motion analysis methods and applications using a consumer video game controller. *Proceedings of the 2007 ACM SIGGRAPH Symposium on Video games*, 2007.

[28] G. D. Hager. A tutorial on visual servo control. *IEEE Transaction on Robotics and Automation, vol. 12, no. 5, pp. 651-670*, 1996.

[29] V. Aidala and S. Hammel. Utilization of modified polar coordinates for bearings-only tracking. *IEEE Transactions on Automatic Control, vol. 28, no. 3, pp. 283–294*, 1983.

[30] R Holmberg. *Design and Development of Powered-Castor Holonomic Mobile Robots*. PhD thesis, Stanford University, 2000.

[31] J. Borenstein. Umbmark: a method for measuring, comparing, and correcting dead-reckoning errors in mobile robots. 1994.

[32] H. Everett and L. Feng J. Borenstein. Navigating mobile robots: Systems and techniques. *AK Peters, Ltd. Natick, MA, USA*, 1996.

[33] J. L. Jones and Anita M. Flynn. *Mobile Robots Inspiration to Implementation*. Wellesley.

[34] D. Kragic. Real-time tracking meets online grasp planning. *Proceedings of IEEE International Conference on Robotics and Automation*, 2001.

[35] E. Malis. 2d 1/2 visual servoing. *IEEE Transaction on Robotics and Automation*, 1999.

[36] S Papert. *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books, 1980.

[37] M Resnick. A study of children and concurrent programming. *Interactive Learning Environments, vol. 1, no. 3, pp. 153-170*, 1991.

[38] F. Ribeiro, I. Moutinho, P. Silva, C. Fraga, and N. Pereira. Three omni-directional wheels control on a mobile robot.

[39] P. Boulanger F. Blais S. F. El-Hakim. A mobile system for indoors 3-d mapping and positioning. *Optical 3-D Measurement Techniques IV. Zurich: Wichmann Press*, 1997.

[40] C. E. Guestrin R. Sukthankar and M. Paskin S. Funiak. Distributed localization of networked cameras. *Fifth Intâl Conf. on Information Processing in Sensor Networks*, 2006.

[41] S. Thrun. Probabilistic algorithms in robotics. *AI Magazine, vol. 21, no. 4, pp. 93–109*, 2000.

[42] Manuela M. Veloso. Entertainment robotics. *Commun. ACM*, 45:59–63, March 2002.

[43] G. Xiang Y.-H. Liu K. Li Y. Shen. Uncalibrated visual servoing of planar robots. *Proceedings of IEEE International Conference on Robotics and Automation*, 2002.

Appendix A

Documentation of the project logic

Documentation of the logical design which is documenting the logical design of the system and the design of SW. This appendix shows the logical architecture implemented

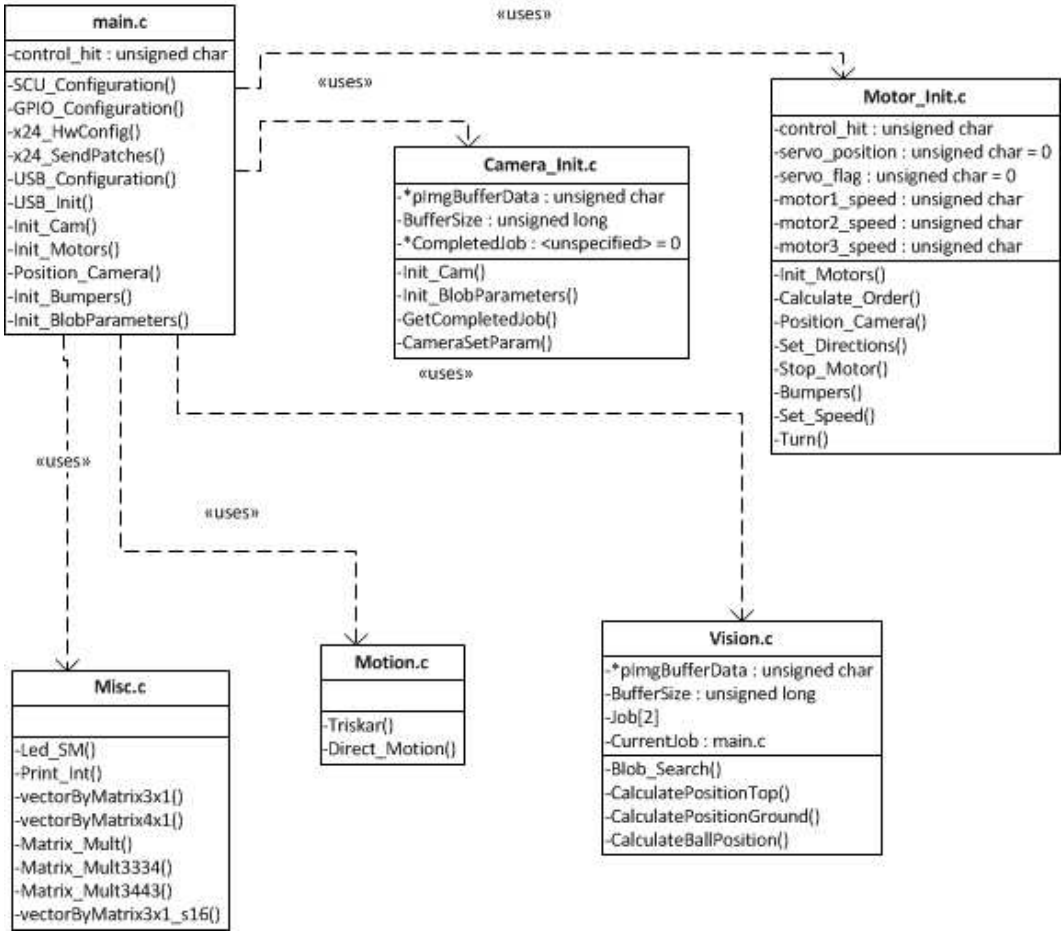

Figure A.1: The class diagram of the most used classes

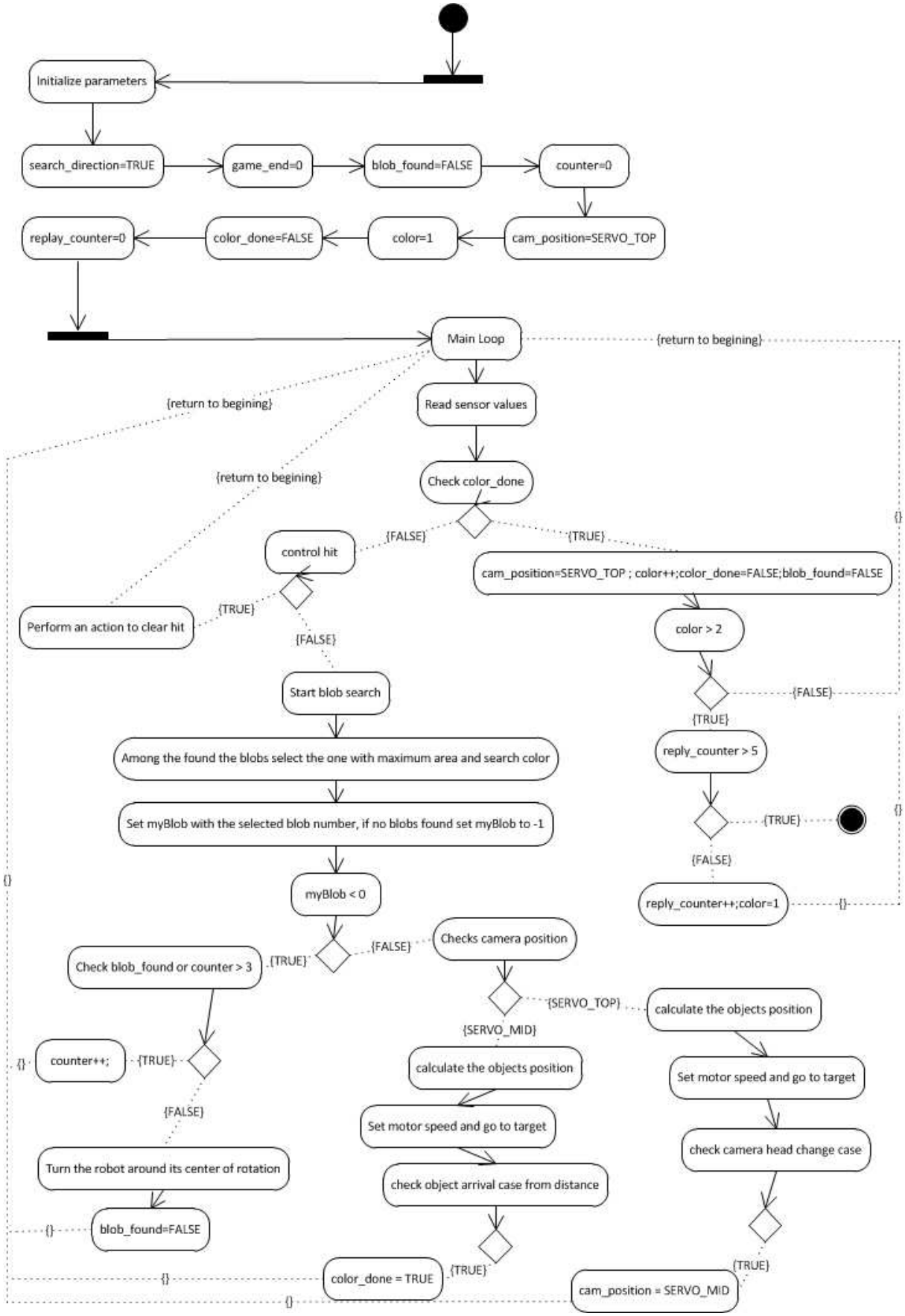

Figure A.2: The flow diagram of the game algorithm

Appendix B

Documentation of the programming

The microprocessor code, the scripts and other helper tools that are used during the implementation, are included here.

B.1 Microprocessor Code

———————main.c———————

```
/******************** (C) COPYRIGHT 2007 STMicroelectronics ********************
* File Name          : main.c
* Author             : AST Robotics group
* Date First Issued  : 11 May 2007 : Version 1.0
* Description        : Main program body
********************************************************************************
* History:
* 28 May 2007 : Version 1.2
* 11 May 2007 : Version 1.0
********************************************************************************
* THE PRESENT SOFTWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.
* AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT,
* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE
* CONTENT OF SUCH SOFTWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING
* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "91x_lib.h"
#include "definitions.h"
#include "CamInt_x24.h"
```

```
#include "utils.h"
#include "math.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
#include "definitionsLib.h"
#include "Misc.h"
#include "Camera_Init.h"
#include "Motor_Init.h"
#include "blobsearch.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
extern u8 control_hit;
/* Private function prototypes -----------------------------------------------*/

void SCU_Configuration(void);
void USB_Configuration(void);
void GPIO_Configuration(void);

/* Private functions ---------------------------------------------------------*/

/*******************************************************************************
* Function Name  : main
* Description    : Main program
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
int main(void) {

#ifdef DEBUG
debug();
#endif

/* Configure the system clocks */
SCU_Configuration();

/* Configure the GPIO ports */
GPIO_Configuration();

/* Configure x24 Camera */
x24_HwConfig(8000000);
x24_SendPatches();
```

```
/* Configure the USB */
USB_Configuration();
USB_Init();

LED_ON(LEDR);
MSDelay(200);
LED_OFF(LEDR);
MSDelay(200);
LED_ON(LEDR);
MSDelay(200);
LED_OFF(LEDR);
MSDelay(200);
LED_ON(LEDR);
MSDelay(200);
LED_OFF(LEDR);

/* To initialize camera parameters*/
Init_Cam();

/* To initialize motors, configure pins and PWM values*/
Init_Motors();
/* To arrange camera head position
* available position SERVO_TOP,SERVO_MID,SERVO_BOTTOM
* */
Position_Camera(SERVO_TOP);
/* To initialize bumpers, and configure pins */
Init_Bumpers();

MSDelay(2000);
/* To define blob color & geometry */
Init_BlobParameters();

/* the initial values */
bool search_direction = TRUE;
u8 game_end = 0;
bool blob_found = FALSE;
bool color_done = FALSE;
u8 cam_position = SERVO_TOP;
u8 counter = 0;
u8 color = 1;
s16 speed[3];
u8 replay_counter = 0;
while (1) {
/*reading the bumpers values */
u8 check_bumpers[6] = { 0 };
Bumpers(check_bumpers);
/*to blink led*/
Led_SM();
/* controlling whether the first target acquired */
```

```
if (color_done == TRUE) {
/* to check whether game_end satified */
Stop_Motor(3);
MSDelay(100);
cam_position = SERVO_TOP;
Position_Camera(SERVO_TOP);
color++;
color_done = FALSE;
blob_found = FALSE;
if (color > 2) {
if (replay_counter < 5) {
replay_counter++;
color = 1;
} else {
Stop_Motor(3);
while (1)
;
}

}

} else { //else of color_1_done
if (control_hit == 1) { //checking collision
//do something
if (check_bumpers[0] == 1
|| check_bumpers[4] == 1
|| check_bumpers[3] == 1
|| check_bumpers[5] == 1)//hit front
{//go back
speed[0] = 1000;
speed[1] = -1000;
speed[2] = 0;
Set_Speed(speed, NORMAL);
MSDelay(100);
} else if (check_bumpers[1] == 1)//hit right
{//go left
speed[0] = -750;
speed[1] = -750;
speed[2] = 1200;
Set_Speed(speed, NORMAL);
MSDelay(100);
} else if (check_bumpers[2] == 1)//hit left
{//go right
speed[0] = 750;
speed[1] = 750;
speed[2] = -1200;
Set_Speed(speed, NORMAL);
MSDelay(100);
}
```

```
#ifdef NO_DEBUG
Print_Int(speed[0],"b1␣motor␣send:");
Print_Int(speed[1],"b2␣motor␣send:");
Print_Int(speed[2],"b3␣motor␣send:");
#endif
} else { //control hit else
#ifdef NO_DEBUG
Print_Int(control_hit,"hit?");
#endif
Blob_Search();
#ifdef NO_DEBUG
Print_Int(BlobCount,"␣blobs␣found...\n");
#endif

TPoint n_point;
u8 k;
s8 myBlob;
s16 maxArea;
maxArea = 0;
myBlob = -1;
/*
* among the available blob the blob with the
* biggest size is selected as the blob
*/
for (k = 0; k < BlobCount; k++) {
if (Blobs[k].ColorSetID == color) {
if (maxArea < Blobs[k].Area) {
maxArea = Blobs[k].Area;
myBlob = k;
}
}
}

//if(BlobCount == 0){
if (myBlob < 0) {
//random search
#ifdef NO_DEBUG
USB_WriteString("Searching␣for␣a␣blob...\n");
#endif
/* in order to prevent continue turning to search for blob
* counter introduced.
*/
if (blob_found == TRUE) {
counter++;
} else if (counter > 3 || blob_found
== FALSE) {
/*search_direction changes the turning side to left to
* right or vice versa
```

```
*/
if (search_direction == TRUE) {
Turn(RIGHT);
} else {
Turn(LEFT);
}
blob_found = FALSE;
}
} else {
/*
* search direction is reversed
*/
search_direction = !search_direction;

/*
* controlling if the blob is with the correct color
*/
blob_found = TRUE;
counter = 0;
n_point.X = Blobs[myBlob].Centroid.X;
n_point.Y
= Blobs[myBlob].OuterBox.Bottom;
s16 position[3];

/*
* for different camera positions different
* calibrations and motion mechanism is used
*/

if (cam_position == SERVO_TOP) {
CalculatePositionTop(n_point,
position);
#ifdef NO_DEBUG
Print_Int(n_point.X,"X␣in␣cam␣coordinates");
Print_Int(n_point.Y,"Y␣in␣cam␣coordinates");
Print_Int(position[0],"X␣in␣real␣coordinates");
Print_Int(position[1],"Y␣in␣real␣coordinates");
#endif
double alpha = 0;
alpha = atan2(position[1],
position[0]);
s8 omega_modifid = alpha * 100;
Triskar(position, omega_modifid,
speed);

if (Abs(position[0]) > 1200) {
speed[2] = speed[2] / 2;
Set_Speed(speed, FAR);
} else if (Abs(position[0]) > 700) {
```

```
Set_Speed(speed, NORMAL);
} else {
cam_position = SERVO_MID;
Position_Camera(SERVO_MID);
}
#ifdef NO_DEBUG
Print_Int(speed[0],"1 motor send:");
Print_Int(speed[1],"2 motor send:");
Print_Int(speed[2],"3 motor send:");
#endif

}//servo_top
else {//servo_mid
CalculatePositionGround(n_point,
position);
double alpha = 0;
alpha = atan2(position[1],
position[0]);
s8 omega_modifid = alpha * 100;
Triskar(position, omega_modifid,
speed);
speed[2] = speed[2] / 2;
Set_Speed(speed, CLOSE);
if (n_point.Y > 115) {
color_done = TRUE;
}
}
}//blob

}//else of control hit
}

}
}

/*******************************************************************************
* Function Name  : SCU_Configuration
* Description    : Configures the system clocks.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SCU_Configuration(void) {
/* Initialize PLL */
SCU_MCLKSourceConfig(SCU_MCLK_OSC);
SCU_FMICLKDivisorConfig(SCU_FMICLK_Div1);
FMI_Config(FMI_READ_WAIT_STATE_2,
FMI_WRITE_WAIT_STATE_0, FMI_PWD_ENABLE,
FMI_LVD_ENABLE, FMI_FREQ_HIGH);
```

```
SCU_RCLKDivisorConfig(SCU_RCLK_Div1);
SCU_HCLKDivisorConfig(SCU_HCLK_Div1);
SCU_PCLKDivisorConfig(SCU_PCLK_Div2);
SCU_BRCLKDivisorConfig(SCU_BRCLK_Div1);
SCU_PLLFactorsConfig(192, 25, 2); /* PLL = 96 MHz */
SCU_PLLCmd(ENABLE); /* PLL Enabled  */
SCU_MCLKSourceConfig(SCU_MCLK_PLL); /* MCLK = PLL   */

SCU_PFQBCCmd(ENABLE);

/* Enable GPIO 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 Clocks */
SCU_APBPeriphClockConfig(__GPIO0, ENABLE);
GPIO_DeInit(GPIO0);
SCU_APBPeriphClockConfig(__GPIO1, ENABLE);
GPIO_DeInit(GPIO1);
SCU_APBPeriphClockConfig(__GPIO2, ENABLE);
GPIO_DeInit(GPIO2);
SCU_APBPeriphClockConfig(__GPIO3, ENABLE);
GPIO_DeInit(GPIO3);
SCU_APBPeriphClockConfig(__GPIO4, ENABLE);
GPIO_DeInit(GPIO4);
SCU_APBPeriphClockConfig(__GPIO5, ENABLE);
GPIO_DeInit(GPIO5);
SCU_APBPeriphClockConfig(__GPIO6, ENABLE);
GPIO_DeInit(GPIO6);
SCU_APBPeriphClockConfig(__GPIO7, ENABLE);
GPIO_DeInit(GPIO7);
SCU_APBPeriphClockConfig(__GPIO8, ENABLE);
GPIO_DeInit(GPIO8);
SCU_APBPeriphClockConfig(__GPIO9, ENABLE);
GPIO_DeInit(GPIO9);

/* Enable VIC clock */
SCU_AHBPeriphClockConfig(__VIC, ENABLE);
VIC_DeInit();

/* Enable WIU clock */
SCU_APBPeriphClockConfig(__WIU, ENABLE);
WIU_DeInit();

/* Enable WIU clock */
SCU_APBPeriphClockConfig(__I2C0, ENABLE);

/* Enable DMA clock */
SCU_AHBPeriphClockConfig(__DMA, ENABLE);
DMA_DeInit();

/* Enable TIM0123 clock */
SCU_APBPeriphClockConfig(__TIM01, ENABLE);
```

```
TIM_DeInit(TIM0);
TIM_DeInit(TIM1);
SCU_APBPeriphClockConfig(__TIM23, ENABLE);
TIM_DeInit(TIM2);
TIM_DeInit(TIM3);
SCU_TIMPresConfig(SCU_TIM01, 4800); // ~10KHz
SCU_TIMExtCLKCmd(SCU_TIM01, DISABLE); // Disable external pin
SCU_TIMPresConfig(SCU_TIM23, 4800); // ~10KHz
SCU_TIMExtCLKCmd(SCU_TIM23, DISABLE); // Disable external pin

SCU_APBPeriphClockConfig(__I2C0, ENABLE);
I2C_DeInit(I2C0);

SCU_AHBPeriphClockConfig(__FMI, ENABLE);
SCU_AHBPeriphReset(__FMI, DISABLE);
}

/*******************************************************************************
* Function Name  : USB_Configuration
* Description    : Configures the USB
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USB_Configuration(void) {
GPIO_InitTypeDef GPIO_InitStructure;

/* USB clock = MCLK/2 = 48MHz */
SCU_USBCLKConfig(SCU_USBCLK_MCLK2);
/* Enable USB clock */
SCU_AHBPeriphClockConfig(__USB, ENABLE);
SCU_AHBPeriphReset(__USB, DISABLE);
SCU_AHBPeriphClockConfig(__USB48M, ENABLE);

/* Configure USB D+ PullUp pin */
GPIO_StructInit(&GPIO_InitStructure);
GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
GPIO_InitStructure.GPIO_Pin = USB_Dp_PullUp_GPIOx_Pin;
GPIO_InitStructure.GPIO_Type = GPIO_Type_OpenCollector;//GPIO_Type_PushPull;
GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
GPIO_Init(USB_Dp_PullUp_GPIO, &GPIO_InitStructure);
USB_Dp_PullUp_OFF();
MSDelay(100);
USB_Dp_PullUp_ON();

USB_endPointConf();

VIC_Config(USBLP_ITLine, VIC_IRQ, USB_Priority);
VIC_ITCmd(USBLP_ITLine, ENABLE);
```

```
}

/*******************************************************************************
* Function Name  : GPIO_Configuration
* Description    : Configures the different GPIO ports.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void GPIO_Configuration(void) {
GPIO_InitTypeDef GPIO_InitStructure;

/* Configure LEDR */
GPIO_StructInit(&GPIO_InitStructure);
GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
GPIO_InitStructure.GPIO_Pin = LEDR_GPIOx_Pin;
GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
GPIO_Init(LEDR_GPIO, &GPIO_InitStructure);

/* Configure MY_GPIO */
GPIO_StructInit(&GPIO_InitStructure);
GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
GPIO_InitStructure.GPIO_Pin = MY_GPIOx_Pin;
GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
GPIO_Init(MY_GPIO, &GPIO_InitStructure);
}

/****************** (C) COPYRIGHT 2007 STMicroelectronics *****END OF FILE****/
```

————Camera_Init.h.————

```
/*
 * Camera_Init.h
 *
 *  Created on: Jan 19, 2011
 *      Author: Administrator
 */

#ifndef CAMERA_INIT_H_
#define CAMERA_INIT_H_

/* Includes ------------------------------------------------------------------*/
#include "91x_lib.h"
#include "CamInt_x24.h"

/* Exported types ------------------------------------------------------------*/
```

```
/* Exported constants --------------------------------------------------------*/

/* Module private variables ---------------------------------------------------*/
extern u8 *pImgBufferData;
extern u32 BufferSize;
extern u8 CamState; // 0:Uniitialized, 1:Running, 2:Paused
extern Tx24_ImgJob *CompletedJob;

/* Exported macro ------------------------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Init_Cam(void);
void GetCompletedJob(void);
void CameraSetParam(u8 ImgFormat, u16 Width, u16 Height,
                u8 Framerate, u8 SensorMode, u8 HalfSysClock,
                u8 JpegCompr, u8 JpegDerat);
void Init_BlobParameters(void);

#endif /* CAMERA_INIT_H_ */
```

Camera_Init.c.

```
/* Standard include ----------------------------------------------------------*/
#include "91x_map.h"
#include "utils.h"
#include "definitions.h"
#include "definitionsLib.h"
#include "CamInt_x24.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
#include "RGBCube.h"
#include "blobsearch.h"
#include "math.h"
#include "Camera_Init.h"

/* Include of other module interface headers ---------------------------------*/
/* Local includes ------------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
#define BUFFERSIZE_DEMO_BS      160*120*2
/* Private define ------------------------------------------------------------*/
#define TimeDiv 250000000
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
const u16 SyncOutPeriods[150] = { 9999, 8332, 7142, 6249,
                5555, 4999, 4545, 4166, 3846, 3571, 3333, 3125,
                2941, 2777, 2631, 2500, 2381, 2272, 2174, 2083,
```

```
                2000, 1923, 1852, 1786, 1724, 1666, 1613, 1562,
                1515, 1470, 1428, 1389, 1351, 1316, 1282, 1250,
                1219, 1190, 1163, 1136, 1111, 1087, 1064, 1042,
                1020, 1000, 980, 961, 943, 926, 909, 893, 877, 862,
                847, 833, 820, 806, 794, 781, 769, 757, 746, 735,
                725, 714, 704, 694, 685, 676, 667, 658, 649, 641,
                633, 625, 617, 610, 602, 595, 588, 581, 575, 568,
                562, 555, 549, 543, 538, 532, 526, 521, 515, 510,
                505, 500, 495, 490, 485, 481, 476, 472, 467, 463,
                459, 454, 450, 446, 442, 439, 435, 431, 427, 424,
                420, 417, 413, 410, 406, 403, 400, 397, 394, 391,
                388, 385, 382, 379, 376, 373, 370, 368, 365, 362,
                360, 357, 355, 352, 350, 347, 345, 342, 340, 338,
                336, 333 };
u8 *pImgBufferData;
u32 BufferSize;
Tx24_ImgJob Job[2], *CompletedJob = 0;
u8 CurrentJob;
u8 CamState = 0; // 0:Uniitialized, 1:Running, 2:Paused
TTimeStamp Tcam1, Tcam2;
vs32 OldErr = 0;
vs32 Ierr = 0, Nerr = 0;

/* Private function prototypes -----------------------------------------------*/

/* Private functions ---------------------------------------------------------*/

void Init_BlobParameters(void) {
        // Define blobsearch constraints
        BlobSearchSetOption(5, 0, 120, 0, 160);
        // Define blobsearch geometry options
        //correct tihs
        BlobSearchSetColParam(1, 30, 2000, 50, 13000, 26, 153);
        BlobSearchSetColParam(2, 30, 2000, 50, 30000, 26, 153);

        //my green
        //day
        //      RGBCube_AddCube(0,6,4,8,0,1,1);

        //cloudly
        //day
        //      RGBCube_AddCube(0,6,4,8,1,3,1);
        //      RGBCube_SubCube(3,5,3,5,1,3,1);
        //end day
        RGBCube_AddCube(0, 6, 4, 6, 2, 3, 1);
        RGBCube_SubCube(3, 5, 2, 4, 1, 3, 1);
        RGBCube_SubCube(5, 7, 4, 6, 2, 4, 1);
        //      RGBCube_SubCube(3,5,4,6,2,4,1);
        //my orange
```

```
        RGBCube_AddCube(6, 8, 3, 5, 1, 3, 2);
        RGBCube_AddCube(5, 7, 1, 3, 0, 1, 2);

        /*night
         RGBCube_AddCube(0,6,4,8,0,2,1);
         RGBCube_SubCube(3,5,3,5,1,3,1);
         night*/
}
void Init_Cam(void) {

        u8 ImgBufferData[2 * BUFFERSIZE_DEMO_BS];
        u8 RGBCubeData[4096];

        BufferSize = BUFFERSIZE_DEMO_BS;

        pImgBufferData = ImgBufferData;
        pRGBCubeData = RGBCubeData;

        RGBCube_Clear();
        BlobSearchInit();
        /*
         * white balance & exposure compensation settings
         */
        x24_WriteReg8(0x1380, 0); //wb
        x24_WriteReg8(0x1080, 2); //ae
        x24_WriteReg16(0x1095, 1000);
        x24_WriteReg16(0x109d, 240);
        x24_WriteRegF900(0x10a1, 1.0);

        u8 ImgFormat = x24_ImageFormat_RGB_565;
        u16 Width = 160;
        u16 Height = 120;
        u8 Framerate = 25;
        u8 SensorMode = x24_SensorMode_SVGA;
        u8 HalfSysClock = 80;
        u8 JpegCompr = 150;
        u8 JpegDerat = 10;
        CameraSetParam(ImgFormat, Width, Height, Framerate,
                SensorMode, HalfSysClock, JpegCompr, JpegDerat);
}

void GetCompletedJob(void) {
        if (Job[CurrentJob].State == x24_StateAcquisitionEnd) {
                CompletedJob = &Job[CurrentJob];
                CurrentJob ^= 1;
        }
}

void CameraSetParam(u8 ImgFormat, u16 Width, u16 Height,
```

```
                u8 Framerate, u8 SensorMode, u8 HalfSysClock,
                u8 JpegCompr, u8 JpegDerat) {
        Tx24_InitTypeDef x24_InitStruct;

        /* Initialize the x24 module */
        x24_StructInit(&x24_InitStruct);
        x24_InitStruct.SensorMode = SensorMode;
        x24_InitStruct.DesiredFrameRate_Num = Framerate * 250;
        x24_InitStruct.DesiredFrameRate_Den = 250;
        x24_InitStruct.ExtClockFreqMhz_Num = 8;
        x24_InitStruct.ExtClockFreqMhz_Den = 1;
        x24_InitStruct.JPEGClockDerate = JpegDerat;
        x24_InitStruct.SyncEnabled = 0;

        x24_InitStruct.ImageWidth = Width;
        x24_InitStruct.ImageHeight = Height;
        x24_InitStruct.ImageFormat = ImgFormat; // x24_ImageFormat_RGB_565 x24_ImageF
        x24_InitStruct.JPEGSqueezeValue = JpegCompr; // use 100 [800x600], 150 [1024x
        x24_InitStruct.SysClock = HalfSysClock * 2.0; // 35MHz; Range: [35:270] MHz
320*240@10Hz RGB_565

        x24_InitStruct.MemBlocksNum = 2;
        x24_InitStruct.ImgBlocksNum = 1;
        x24_InitStruct.ImgBlockSize = BufferSize;
        x24_InitStruct.pDataMemory = pImgBufferData;

        x24_InitStruct.ExposureCompensation
                        = x24_biExposureCompensation;

        x24_Init(&x24_InitStruct);

        Job[0].pNext = &Job[1];
        Job[1].pNext = &Job[0];
        CurrentJob = 0;
        x24_GrabFrameStart(&Job[0]);

}
```

————Motor_Init.h.————

```
/*
 * Motor_Init.h
 *
 *  Created on: Jan 19, 2011
 *      Author: Administrator
 */

#ifndef MOTOR_INIT_H_
#define MOTOR_INIT_H_
```

```
/* Includes ------------------------------------------------------------------*/

/* Exported types ------------------------------------------------------------*/

/* Exported constants --------------------------------------------------------*/

/* Module private variables ---------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Init_Motors(void);
void Calculate_Order(s16* speed, u8 size);
void Position_Camera(u8 position);
void Set_Directions(u8 Motor1, u8 Motor2, u8 Motor3);
void Stop_Motor(u8 Motor);
void Bumpers(u8 bumper[6]);
void Set_Speed(s16* speed, u8 mod);
void Stop_Servo(void);
void Turn(u8 direction);

#endif /* MOTOR_INIT_H_ */
```

Motor_Init.c

```
/* Standard include ----------------------------------------------------------*/
#include "91x_map.h"
#include "utils.h"
#include "definitions.h"
#include "definitionsLib.h"
#include "CamInt_x24.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
#include "math.h"
#include "Motor_Init.h"

/* Include of other module interface headers ---------------------------------*/
/* Local includes ------------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
u8 control_hit;
u8 servo_position = 0;
u8 servo_flag = 0;

u8 motor1_speed;
```

```
u8 motor2_speed;
u8 motor3_speed;

/* Private function prototypes -----------------------------------------------*/
void Init_Motors(void);
void Calculate_Order(s16* speed, u8 size);
void Position_Camera(u8 position);
void Set_Directions(u8 Motor1, u8 Motor2, u8 Motor3);
void Stop_Motor(u8 Motor);
void Bumpers(u8 bumper[6]);
void Set_Speed(s16* speed, u8 mod);
void Stop_Servo(void);
void Turn(u8 direction);
/* Private functions ---------------------------------------------------------*/

void Set_Speed(s16* speed, u8 mod) {

        u8 direction[NUM_MOTORS];
        u8 k;

        for (k = 0; k < NUM_MOTORS; k++) {
                direction[k] = (speed[k] > 0) ? 1 : 0, speed[k]
                                =Abs(speed[k]);

        }

        for (k = 0; k < NUM_MOTORS; k++) {

                if (speed[k] == 0) {
                        Stop_Motor(k);
                }
                if (speed[k] > 2500)
                        speed[k] = 2500;
        }

        for (k = 0; k < NUM_MOTORS; k++) {
                if (speed[k] != 0)
                        speed[k] = (speed[k] / 75 * mod) + 90;
        }

        Set_Directions(direction[0], direction[1], direction[2]);

        motor1_speed = (u8) speed[0];
        motor2_speed = (u8) speed[1];
        motor3_speed = (u8) speed[2];
#ifdef NO_DEBUG
        Print_Int(motor1_speed,"m1␣applied");
        Print_Int(motor2_speed,"m2␣applied");
        Print_Int(motor3_speed,"m3␣applied");
```

```
#endif
}

void Stop_Motor(u8 Motor) {
        switch (Motor) {
        case 0: //stop 1
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_B, Bit_RESET);
                break;
        case 1: //stop 2
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_B, Bit_RESET);
                break;
        case 2: //stop 3
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_B, Bit_RESET);
                break;
        case 3: //stop all
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_B, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_B, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_B, Bit_RESET);
                break;
        }
}

void Set_Directions(u8 Motor1, u8 Motor2, u8 Motor3) {
        if (Motor2 == FORWARD) {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_A, Bit_SET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_B, Bit_RESET);
        } else {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_A, Bit_RESET);
```

```
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR2_DIRECTION_B, Bit_SET);
        }
        if (Motor1 == FORWARD) {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_A, Bit_SET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_B, Bit_RESET);
        } else {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR1_DIRECTION_B, Bit_SET);
        }
        if (Motor3 == FORWARD) {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_A, Bit_SET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_B, Bit_RESET);
        } else {
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_A, Bit_RESET);
                GPIO_WriteBit(MOTOR_DIRECTION_GPIO,
                        MOTOR3_DIRECTION_B, Bit_SET);
        }

}

void Calculate_Order(s16* speed, u8 size) {

        u8 pass = 1;
        u8 sorted = 0;
        u8 i;
        while ((!sorted) && (pass < size)) {
                sorted = 1;
                for (i = 0; i < size - pass; i++) {
                        if (speed[i] > speed[i + 1]) {
                                int temp = speed[i];
                                speed[i] = speed[i + 1];
                                speed[i + 1] = temp;
                                sorted = 0;

                        }
                }
                pass++;
        }
}
void Stop_Servo() {
        servo_flag = STOP_SERVO;
```

```
}

void Init_Motors() {
        GPIO_InitTypeDef GPIO_InitStructure;

        GPIO_StructInit(&GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
        GPIO_InitStructure.GPIO_Pin = MOTOR1_Pin | MOTOR2_Pin
                        | MOTOR3_Pin;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
        GPIO_Init(MOTOR_GPIO, &GPIO_InitStructure);

        /* Config Servo PIN */
        GPIO_StructInit(&GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
        GPIO_InitStructure.GPIO_Pin = SERVO_Pin;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
        GPIO_Init(SERVO_GPIO, &GPIO_InitStructure);

        GPIO_StructInit(&GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Direction = GPIO_PinOutput;
        GPIO_InitStructure.GPIO_Pin = MOTOR1_DIRECTION_A
                        | MOTOR1_DIRECTION_B | MOTOR2_DIRECTION_A
                        | MOTOR2_DIRECTION_B | MOTOR3_DIRECTION_A
                        | MOTOR3_DIRECTION_B;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_InitStructure.GPIO_Alternate = GPIO_OutputAlt1;
        GPIO_Init(MOTOR_DIRECTION_GPIO, &GPIO_InitStructure);

        TIM_InitTypeDef TIM_InitStructure;

        TIM_StructInit(&TIM_InitStructure);
        TIM_InitStructure.TIM_Mode = TIM_OCM_CHANNEL_12; //for both use 12
        TIM_InitStructure.TIM_OC1_Modes = TIM_TIMING;
        TIM_InitStructure.TIM_OC2_Modes = TIM_TIMING;
        TIM_InitStructure.TIM_Clock_Source = TIM_CLK_APB;
        TIM_InitStructure.TIM_Prescaler = 47; //127
        TIM_InitStructure.TIM_Pulse_Length_1 = 50;
        TIM_InitStructure.TIM_Pulse_Length_2 = 200;
        TIM_InitStructure.TIM_Pulse_Level_1 = TIM_HIGH;
        TIM_InitStructure.TIM_Pulse_Level_2 = TIM_HIGH;
        TIM_InitStructure.TIM_Period_Level = TIM_LOW;

        TIM_Init(TIM0, &TIM_InitStructure);
        TIM_Init(TIM1, &TIM_InitStructure);
        TIM_Init(TIM3, &TIM_InitStructure);
```

```
        /*Enable TIM0 Output Compare1 interrupt*/
        TIM_ITConfig(TIM0, TIM_IT_TO | TIM_IT_OC1 | TIM_IT_OC2,
                ENABLE);
        /*Enable TIM1 Output Compare1 interrupt*/
        TIM_ITConfig(TIM1, TIM_IT_TO | TIM_IT_OC1 | TIM_IT_OC2,
                ENABLE);
        /*Enable TIM3 Output Compare1 interrupt*/
        TIM_ITConfig(TIM3, TIM_IT_TO | TIM_IT_OC1 | TIM_IT_OC2,
                ENABLE);

        VIC_Config(TIM0_ITLine, VIC_IRQ, 9);
        VIC_ITCmd(TIM0_ITLine, ENABLE);

        VIC_Config(TIM1_ITLine, VIC_IRQ, 10);
        VIC_ITCmd(TIM1_ITLine, ENABLE);

        VIC_Config(TIM3_ITLine, VIC_IRQ, 11);
        VIC_ITCmd(TIM3_ITLine, ENABLE);

        /*Start*/
        TIM_CounterCmd(TIM0, TIM_START);
        TIM_CounterCmd(TIM1, TIM_START);
        TIM_CounterCmd(TIM3, TIM_START);

        u8 i;
        s16 speed[3];
        for (i = 0; i < NUM_MOTORS; i++) {
                speed[i] = 0;
        }

        Set_Speed(speed, CLOSE);
}

void Position_Camera(u8 position) {

        servo_flag = 0;
        servo_position = position;

}
void Init_Bumpers() {
        GPIO_InitTypeDef GPIO_InitStructure;

        GPIO_StructInit(&GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Direction = GPIO_PinInput;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_Init(GPIO5, &GPIO_InitStructure);

        GPIO_StructInit(&GPIO_InitStructure);
```

```
        GPIO_InitStructure.GPIO_Direction = GPIO_PinInput;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_7;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_Init(GPIO1, &GPIO_InitStructure);

        GPIO_StructInit(&GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Direction = GPIO_PinInput;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Type = GPIO_Type_PushPull;
        GPIO_Init(GPIO4, &GPIO_InitStructure);

}
void Bumpers(u8 bumper[6]) {
        control_hit = 0;

        if (GPIO_ReadBit(GPIO5, BUMPER_FRONT_LEFT) == Bit_SET) {
                bumper[0] = 1;
        }
        if (GPIO_ReadBit(GPIO4, BUMPER_FRONT_RIGHT) == Bit_SET) {
                bumper[4] = 1;
        }
        if (GPIO_ReadBit(GPIO1, BUMPER_LEFT) == Bit_SET) {
                bumper[2] = 1;
        }
        if (GPIO_ReadBit(GPIO5, BUMPER_RIGHT) == Bit_SET) {
                bumper[1] = 1;
        }
        if (GPIO_ReadBit(GPIO1, BUMPER_BACK_LEFT) == Bit_SET) {
                bumper[3] = 1;
        }

        if (GPIO_ReadBit(GPIO4, BUMPER_BACK_RIGHT) == Bit_SET) {
                bumper[5] = 1;
        }
        u8 i;

        for (i = 0; i < 6; i++) {
                if (bumper[i] == 1) {
                        control_hit = 1;
#ifdef NO_DEBUG
                        Print_Int(i,"no");
                        Print_Int(bumper[i],"Bumper");
#endif
                }
        }
}

void Turn(u8 direction) {
```

```
        s16 left_speed[3] = { 550, 550, 550 };
        s16 right_speed[3] = { -550, -550, -550 };

        if (direction == LEFT) {
                Set_Speed(left_speed, NORMAL);

        } else {
                Set_Speed(right_speed, NORMAL);
        }
        MSDelay(100);
        Stop_Motor(3);
        MSDelay(100);
}
```

———Vision.h.———

```
/*
 * Vision.h
 *
 *  Created on: Jan 19, 2011
 *      Author: Administrator
 */

#ifndef VISION_H_
#define VISION_H_
/* Includes ------------------------------------------------------------------*/
#include "91x_lib.h"
#include "CamInt_x24.h"

/* Exported types ------------------------------------------------------------*/

/* Exported constants --------------------------------------------------------*/

/* Module private variables ---------------------------------------------------*/
extern u8 *pImgBufferData;
extern u32 BufferSize;
/* Exported macro ------------------------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void CalculateBallPosition(u8 X, u8 Y, u8 Dpx,
                double result[4]);
void CalculatePositionTop(TPoint point, s16 result[3]);
void CalculatePositionGround(TPoint point, s16 result[3]);
void Blob_Search(void);

#endif /* VISION_H_ */
```

———Vision.c.———

```
/* Standard include ----------------------------------------------------------*/
#include "91x_map.h"
#include "utils.h"
#include "definitions.h"
#include "definitionsLib.h"
#include "CamInt_x24.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
#include "RGBCube.h"
#include "blobsearch.h"
#include "math.h"
#include "Misc.h"
#include "Vision.h"
#include "Camera_Init.h"
#include "Motion.h"

/* Include of other module interface headers ---------------------------------*/
/* Local includes ------------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
//#define PRINT_ON

/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
u8 *pImgBufferData;
u32 BufferSize;
double BlobSample[10][2];
u8 index = 0;
extern Tx24_ImgJob Job[2];
extern CurrentJob;
extern u8 control_hit;
/* Private function prototypes -----------------------------------------------*/
//see header
/* Private functions ---------------------------------------------------------*/
void Blob_Search(void) {
        if (Job[CurrentJob].State == x24_StateAcquisitionEnd) { // Image is acquired
                MY_ON();
                CompletedJob = &Job[CurrentJob];
                BlobSearch((u16*) CompletedJob->pFirstBlock->pData);
                MY_OFF();
                CurrentJob ^= 1;
        }
}

void CalculatePositionTop(TPoint point, s16 result[3]) {
        //Homography for servo_top
```

```
        double K[3][3] = { { 191.7146, 0, 80.3591 }, { 0,
                        191.2730, 61.2765 }, { 0, 0, 1 } },
                        T_cr[3][4] = { { -0.0755, -0.9948, 0.0683,
                                        -8.6194 }, { -0.1867, -0.0531, -0.9810,
                                        271.8015 }, { 0.9795, -0.0868, -0.1817,
                                        -72.3125 } }, C[3][4] = {
                                        { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 0,
                                                0, 0 } },
                        H[3][3] = { { 0, 0, 0 }, { 0, 0, 0 },
                                        { 0, 0, 0 } },
                        InvH[3][3] = { { 0, 0, 0 }, { 0, 0, 0 }, { 0,
                                        0, 0 } }, t_[4][3] = { { 1, 0, 0 }, {
                                        0, 1, 0 }, { 0, 0, 0 }, { 0, 0, 1 } };

        Matrix_Mult3334(K, T_cr, C);

        Matrix_Mult3443(C, t_, H);

        Inverse(H, InvH);

        double position[3] = { 0, 0, 0 };
        double image_point[3] = { point.X, point.Y, 1 };

        vectorByMatrix3x1(InvH, image_point, position);

        result[0] = (s16) (position[0] / position[2]);
        result[1] = (s16) (position[1] / position[2]);
        result[2] = (s16) (position[2] / position[2]);

}
void CalculatePositionGround(TPoint point, s16 result[3]) {
        //Homography for servo_top
        double K[3][3] = { { 191.7146, 0, 80.3591 }, { 0,
                        191.2730, 61.2765 }, { 0, 0, 1 } },
                        T_cr[3][4] = { { -0.0160, -0.9987, 0.0478,
                                        -0.6271 }, { -0.6634, -0.0252, -0.7478,
                                        245.6112 }, { 0.7481, -0.0437, -0.6622,
                                        44.2694 } }, C[3][4] = {
                                        { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 0, 0,
                                                0, 0 } },
                        H[3][3] = { { 0, 0, 0 }, { 0, 0, 0 },
                                        { 0, 0, 0 } },
                        InvH[3][3] = { { 0, 0, 0 }, { 0, 0, 0 }, { 0,
                                        0, 0 } }, t_[4][3] = { { 1, 0, 0 }, {
                                        0, 1, 0 }, { 0, 0, 0 }, { 0, 0, 1 } };

        Matrix_Mult3334(K, T_cr, C);

        Matrix_Mult3443(C, t_, H);
```

```
        Inverse(H, InvH);

        double position[3] = { 0, 0, 0 };
        double image_point[3] = { point.X, point.Y, 1 };

        vectorByMatrix3x1(InvH, image_point, position);

        result[0] = (s16) (position[0] / position[2]);
        result[1] = (s16) (position[1] / position[2]);
        result[2] = (s16) (position[2] / position[2]);

}

void CalculateBallPosition(u8 X, u8 Y, u8 Dpx,
                double result[4]) {

        double image_point[3] = { X, Y, 1 };
        double inv_K[3][3] = { { 0.005216086284917, 0,
                        -0.419160051539105 }, { 0, 0.005228129700905,
                        -0.320361489617536 }, { 0, 0, 1 } };

        double position[3] = { 0, 0, 0 };

        vectorByMatrix3x1(inv_K, image_point, position);

        double norm;
        norm = sqrt((position[0] * position[0]) + (position[1]
                        * position[1]) + (position[2] * position[2]));

        position[0] = position[0] / norm;
        position[1] = position[1] / norm;
        position[2] = position[2] / norm;

        int X_axis = 300;
        int Y_axis = -150;
        int Dreal = 62; //red ball w 6.2 cm diam

        double fstar = 160.9969;

        double Pc[4] = { (fstar * Dreal / Dpx) * position[0],
                        (fstar * Dreal / Dpx) * position[1], (fstar
                                        * Dreal / Dpx) * position[2], 1 };

        double T_wr[4][4] = { { 1, 0, 0, X_axis }, { 0, 1, 0,
                        Y_axis }, { 0, 0, 1, 0 }, { 0, 0, 0, 1 } };

        double T_wc[4][4] = { { 0.0012, -0.3587, 0.9334,
                        -350.8669 }, { -1.0000, -0.0039, -0.0002,
```

```
                        140.5637 }, { 0.0037, -0.9334, -0.3587,
                        203.8752 }, { 0, 0, 0, 1.0000 } };

        double inv_T[4][4] = { 0, 0, 0, 0 };
        Matrix_Mult(T_wr, T_wc, inv_T);

        vectorByMatrix4x1(inv_T, Pc, result);
}
```

———Motion.h.———

```
/*
 * Motion.h
 *
 *  Created on: Jan 19, 2011
 *      Author: Administrator
 */

#ifndef MOTION_H_
#define MOTION_H_

/* Includes ------------------------------------------------------------------*/

/* Exported types ------------------------------------------------------------*/

/* Exported constants --------------------------------------------------------*/

/* Module private variables ---------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Triskar(s16* destination, s8 omega, s16* vt);
void Direct_Motion(s16* destination, s8 angle, s16* speed);

#endif /* MOTION_H_ */
```

———Motion.c.———

```
/* Standard include ----------------------------------------------------------*/
#include "91x_map.h"
#include "utils.h"
#include "definitions.h"
#include "definitionsLib.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
```

```
#include "math.h"
#include "Misc.h"

/* Include of other module interface headers ----------------------------------*/
/* Local includes -------------------------------------------------------------*/
/* Private typedef ------------------------------------------------------------*/
/* Private define -------------------------------------------------------------*/
/* Private macro --------------------------------------------------------------*/
/* Private variables ----------------------------------------------------------*/
/* Private function prototypes ------------------------------------------------*/
void Triskar(s16* destination, s8 omega, s16* vt);
void Direct_Motion(s16* destination, s8 angle, s16* speed);
/* Private functions ----------------------------------------------------------*/

void Triskar(s16* destination, s8 omega, s16* vt) {
        s16 frontal_speed = destination[0];
        s16 lateral_speed = destination[1];
        double alpha = 0;
        s16 velocity[2];
        velocity[0] = frontal_speed * cos(-alpha)
                        - lateral_speed * sin(-alpha);
        velocity[1] = frontal_speed * sin(-alpha)
                        + lateral_speed * cos(-alpha);

#ifdef NO_DEBUG
        USB_WriteString("Calculating␣Motion␣\n");
        Print_Int(velocity[0],"frontal");
        Print_Int(velocity[1],"lateral");
#endif

        u8 R_robot = 250;
        s16 v_F = velocity[0];
        s16 v_L = velocity[1];
        double d_cosA = 0.8660;
        double d_sinA = 0.5000;
        u8 cosA = d_cosA * 100;
        u8 sinA = d_sinA * 100;
        s16 v[3] = { v_F, v_L, (omega * R_robot) / 100 };
#ifdef NO_DEBUG
        //print velocity vector
        Print_Int(v[0],"v0");
        Print_Int(v[1],"v1");
        Print_Int(v[2],"v2");

#endif
        u8 k, l;

        s16 MF[3][3] = { { -cosA, sinA, -100 }, { cosA, sinA,
                        -100 }, { 0, -100, -100 } };
```

```
#ifdef NO_DEBUG
        //print MF

        for(k=0; k<3;k++) {
                USB_WriteString("\n");
                for(l=0; l<3;l++) {
                        Print_Int(MF[k][l],"MF[][]");
                }
        }
#endif

        vectorByMatrix3x1_s16(MF, v, vt);
#ifdef NO_DEBUG
        //motor speeds found
        Print_Int(vt[0],"vt0");
        Print_Int(vt[1],"vt1");
        Print_Int(vt[2],"vt2");

#endif

}
void Direct_Motion(s16* destination, s8 angle, s16* speed) {
        s16 frontal_speed = destination[0];
        s16 lateral_speed = destination[1];
        double alpha = 0;
        s16 velocity[2];
        velocity[0] = frontal_speed * cos(-alpha)
                        - lateral_speed * sin(-alpha);
        velocity[1] = frontal_speed * sin(-alpha)
                        + lateral_speed * cos(-alpha);

#ifdef NO_DEBUG
        USB_WriteString("Calculating␣Motion␣\n");
        Print_Int(velocity[0],"frontal");
        Print_Int(velocity[1],"lateral");
#endif

        Triskar(velocity, angle, speed);

#ifdef NO_DEBUG
        Print_Int(speed[0],"speed0");
        Print_Int(speed[1],"speed1");
        Print_Int(speed[2],"speed2");
#endif

}
```

```
/*
 * Motion.h
 *
 *  Created on: Jan 19, 2011
 *      Author: Administrator
 */

#ifndef MOTION_H_
#define MOTION_H_

/* Includes ------------------------------------------------------------------*/

/* Exported types ------------------------------------------------------------*/

/* Exported constants --------------------------------------------------------*/

/* Module private variables ---------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */
void Triskar(s16* destination, s8 omega, s16* vt);
void Direct_Motion(s16* destination, s8 angle, s16* speed);

#endif /* MOTION_H_ */
```

———Misc.c.———

```
/* Standard include ----------------------------------------------------------*/
#include "91x_map.h"
#include "utils.h"
#include "definitions.h"
#include "definitionsLib.h"
#include "CamInt_x24.h"
#include "usb_lib.h"
#include "usb_conf.h"
#include "usb_prop.h"
#include "usb_pwr.h"
#include "usb_config.h"
#include "RGBCube.h"
#include "blobsearch.h"
#include "math.h"

/* Include of other module interface headers ---------------------------------*/
/* Local includes ------------------------------------------------------------*/
/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
```

```
void Led_SM(void);
void Print_Int(int a, u8 * str);
void vectorByMatrix3x1(double A[][3], double x[3],
                double b[3]);
void vectorByMatrix4x1(double A[][4], double x[4],
                double b[4]);
void Matrix_Mult(double a1[][4], double a2[][4],
                double a3[][4]);
void Matrix_Mult3334(double a1[][3], double a2[][4],
                double a3[][4]);
void Matrix_Mult3443(double a1[][4], double a2[][3],
                double a3[][3]);
void Inverse(double A[][3], double X[][3]);
void vectorByMatrix3x1_s16(s16 A[][3], s16 x[3], s16 b[3]);
/* Private functions ---------------------------------------------------------*/

void Led_SM(void) {
        static int Count = 0;

        Count++;
        if (Count < 50000) {
                LED_ON(LEDR);
        } else {
                LED_OFF(LEDR);
                if (Count > 60000) {
                        Count = 0;
                }
        }
}

void Print_Int(int a, u8 * str) {
        u8 myStr[10];
        USB_WriteString(str);
        USB_WriteString(":");
        Num2String(a, myStr);
        USB_WriteString(myStr);
        USB_WriteString("\r\n");
}
void vectorByMatrix3x1(double A[][3], double x[3],
                double b[3]) {

        b[0] = A[0][0] * x[0] + A[0][1] * x[1] + A[0][2] * x[2];
        b[1] = A[1][0] * x[0] + A[1][1] * x[1] + A[1][2] * x[2];
        b[2] = A[2][0] * x[0] + A[2][1] * x[1] + A[2][2] * x[2];

}
void vectorByMatrix3x1_s16(s16 A[][3], s16 x[3], s16 b[3]) {
        b[0] = A[0][0] * x[0] / 100 + A[0][1] * x[1] / 100
                        + A[0][2] * x[2] / 100;
```

```
        b[1] = A[1][0] * x[0] / 100 + A[1][1] * x[1] / 100
                        + A[1][2] * x[2] / 100;
        b[2] = A[2][0] * x[0] / 100 + A[2][1] * x[1] / 100
                        + A[2][2] * x[2] / 100;

}
void vectorByMatrix4x1(double A[][4], double x[4],
                double b[4]) {

        b[0] = A[0][0] * x[0] + A[0][1] * x[1] + A[0][2] * x[2]
                        + A[0][3] * x[3];
        b[1] = A[1][0] * x[0] + A[1][1] * x[1] + A[1][2] * x[2]
                        + A[1][3] * x[3];
        b[2] = A[2][0] * x[0] + A[2][1] * x[1] + A[2][2] * x[2]
                        + A[2][3] * x[3];
        b[3] = A[3][0] * x[0] + A[3][1] * x[1] + A[3][2] * x[2]
                        + A[3][3] * x[3];

}
void Matrix_Mult(double a1[][4], double a2[][4],
                double a3[][4]) {
        int i = 0;
        int j = 0;
        int k = 0;
        int a = 4;
        int b = 4;
        int c = 4;

        for (i = 0; i < a; i++)
                for (j = 0; j < b; j++)
                        for (k = 0; k < c; k++)
                                a3[i][j] += a1[i][k] * a2[k][j];
}
void Matrix_Mult3334(double a1[][3], double a2[][4],
                double a3[][4]) {
        int i = 0;
        int j = 0;
        int k = 0;
        for (i = 0; i < 3; i++)
                for (j = 0; j < 4; j++)
                        for (k = 0; k < 3; k++)
                                a3[i][j] += a1[i][k] * a2[k][j];
}

void Matrix_Mult3443(double a1[][4], double a2[][3],
                double a3[][3]) {
        int i = 0;
        int j = 0;
        int k = 0;
```

```
        for (i = 0; i < 3; i++)
                for (j = 0; j < 3; j++)
                        for (k = 0; k < 4; k++)
                                a3[i][j] += a1[i][k] * a2[k][j];
}
void Inverse(double A[][3], double X[][3]) {
        float B[3][3];//the transpose of a matrix A
        float C[3][3];//the adjunct matrix of transpose of a matrix A not adjunct of
        int i, j;
        float x, n = 0;//n is the determinant of A


        for (i = 0, j = 0; j < 3; j++) {
                if (j == 2)
                        n += A[i][j] * A[i + 1][0] * A[i + 2][1];
                else if (j == 1)
                        n += A[i][j] * A[i + 1][j + 1] * A[i + 2][0];
                else
                        n += A[i][j] * A[i + 1][j + 1]
                                        * A[i + 2][j + 2];
        }
        for (i = 2, j = 0; j < 3; j++) {
                if (j == 2)
                        n -= A[i][j] * A[i - 1][0] * A[i - 2][1];
                else if (j == 1)
                        n -= A[i][j] * A[i - 1][j + 1] * A[i - 2][0];
                else
                        n -= A[i][j] * A[i - 1][j + 1]
                                        * A[i - 2][j + 2];
        }

        if (n != 0)
                x = 1.0 / n;
        else {
        }
        for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++) {

                        B[i][j] = A[j][i];

                }
        }

        C[0][0] = B[1][1] * B[2][2] - (B[2][1] * B[1][2]);
        C[0][1] = (-1) * (B[1][0] * B[2][2] - (B[2][0]
                        * B[1][2]));
        C[0][2] = B[1][0] * B[2][1] - (B[2][0] * B[1][1]);

        C[1][0] = (-1)
```

```
                        * (B[0][1] * B[2][2] - B[2][1] * B[0][2]);
        C[1][1] = B[0][0] * B[2][2] - B[2][0] * B[0][2];
        C[1][2] = (-1)
                        * (B[0][0] * B[2][1] - B[2][0] * B[0][1]);

        C[2][0] = B[0][1] * B[1][2] - B[1][1] * B[0][2];
        C[2][1] = (-1)
                        * (B[0][0] * B[1][2] - B[1][0] * B[0][2]);
        C[2][2] = B[0][0] * B[1][1] - B[1][0] * B[0][1];

        for (i = 0; i < 3; i++) {
                for (j = 0; j < 3; j++) {
                        X[i][j] = C[i][j] * x;

                }
        }

}
```

————definitions.h.————

```
/******************** (C) COPYRIGHT 2007 STMicroelectronics ********************
 * File Name          : definitions.h
 * Author             : AST Robotics group
 * Date First Issued  : 11 May 2007 : Version 1.0
 * Description        : generic definitions and pin assignments file for Dongle.
 ********************************************************************************
 * History:
 * 28 May 2007 : Version 1.2
 * 11 May 2007 : Version 1.0
 ********************************************************************************
 THE PRESENT SOFTWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS WITH
 CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.
 AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT, INDIRECT
 OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE CONTENT
 OF SUCH SOFTWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING INFORMATION
 CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
 *******************************************************************************/

/* Define to prevent recursive inclusion ------------------------------------ */
#ifndef __DEFINITIONS_H
#define __DEFINITIONS_H

// *--------------*--------------*------------*
// | UART2_RxD    | N.A.         | UART2_RxD  |
// | UART2_TxD    | N.A.         | UART2_TxD  |
// | I2C_SCL      | P3.4         | SW_I2C_SCL |
// | I2C_SDA      | P3.5         | SW_I2C_SDA |
// | CAM_CE       | P3.6         | GPIO_OUT   |
```

```
// | CAM_PCLK     | P3.0         | Ext_Dma_In |
// | CAM_VSYNC    | P3.2         | EXT_INT_2  |
// | CAM_CLK_IN   | P3.7         | T1_PWM_OUT |
// | CAM_BUS      | P9.ALL       | GPIO_IN    |
// | LED_R        | P6.0         | GPIO_OUT   |
// | SW_1234      | P7.0123      | GPIO_IN    |
// | USB_Dp       | P7.5         | GPIO_OUT   |
// *--------------*--------------*------------*

//#define RVS_MB
#define STL_MB
//#define NO_DEBUG

#define PresentationStringConst "STLCam\nV:0.41\nAST-Robotics\nSTMicroelectronics\n"

#define TMsg_EOF     0xF0
#define TMsg_BS      0xF1
#define TMsg_BS_EOF  0xF2
#define TMsg_MaxLen  128

#define DEV_ADDR     0x32

#define USB_Dp_PullUp_GPIO GPIO0
#define USB_Dp_PullUp_GPIOx_Pin GPIO_Pin_1

#define Cam724
//IRQ Priority    // 0 Highest, 15 Lowest
//#define SERVO_TIM_ITPriority      0
//#define x24_DMA_ITPriority        1
//#define x24_VSYNC_INT_ITPriority  2
//#define x24_SYNC_TIM_IPriority    3
#define USB_Priority              5
/*
 #define MOTOR1_Priority                          9
 #define MOTOR2_Priority                          10
 #define MOTOR3_Priority                          11
 */

#ifdef STL_MB
#define LEDR_GPIO GPIO3
#define LEDR_GPIOx_Pin GPIO_Pin_7
#endif

#define DE_GPIO GPIO4
#define DE_GPIOx_Pin GPIO_Pin_3

#define x24_VSYNC_GPIO GPIO7
#define x24_VSYNC_GPIOx_Pin GPIO_Pin_7
#define x24_VSYNC_INT_WIU_Line ((u32)WIU_Line31)
```

```
#define x24_VSYNC_INT_WIU_Line_N 31
#define x24_VSYNC_INT_ITx_LINE EXTIT3_ITLine

#define MY_GPIO GPIO3
#define MY_GPIOx_Pin GPIO_Pin_2
#define MY MY_GPIO->DR[MY_GPIOx_Pin<<2]
#define MY_ON() MY=0xFF
#define MY_OFF() MY=0x00

#define LEDR LEDR_GPIO->DR[LEDR_GPIOx_Pin<<2]
#define USB_Dp_PullUp USB_Dp_PullUp_GPIO->DR[USB_Dp_PullUp_GPIOx_Pin<<2]

#define DE DE_GPIO->DR[DE_GPIOx_Pin<<2]

#define LED_ON(Led) Led=0xFF;
#define LED_OFF(Led) Led=0x00;
#define LED_TOGGLE(Led) Led=~Led;
#define DE_ON(De) De=0xFF;
#define DE_OFF(De) De=0x00;
#define IS_BTN_PRESSED(Btn) (Btn==0)
#define WAIT_BTN(Btn) { while(!IS_BTN_PRESSED(Btn)); while(IS_BTN_PRESSED(Btn));}
#define USB_Dp_PullUp_OFF(); USB_Dp_PullUp=0xFF;
#define USB_Dp_PullUp_ON(); USB_Dp_PullUp=0x00;

//define motor direction
#define FORWARD 1
#define REVERSE 0

//define servo position
#define SERVO_TOP 0
#define SERVO_MID 1
#define SERVO_BOTTOM 2
#define STOP_SERVO 1
// the servo pins on micro
#define SERVO_GPIO                              GPIO1
#define SERVO_Pin                               GPIO_Pin_1
//motor pins
#define NUM_MOTORS                              3
#define MOTOR_GPIO                              GPIO1
#define MOTOR1_Pin                              GPIO_Pin_6
#define MOTOR2_Pin                              GPIO_Pin_4
#define MOTOR3_Pin                              GPIO_Pin_3
#define MOTOR_DIRECTION_GPIO    GPIO0
//X for the first input which is INA
#define MOTOR1_DIRECTION_A              GPIO_Pin_4
#define MOTOR1_DIRECTION_B              GPIO_Pin_5
#define MOTOR2_DIRECTION_A              GPIO_Pin_2
#define MOTOR2_DIRECTION_B              GPIO_Pin_3
#define MOTOR3_DIRECTION_A              GPIO_Pin_6
```

```
#define MOTOR3_DIRECTION_B              GPIO_Pin_7

//bumper pins
#define BUMPER1                                 GPIO5
#define BUMPER2                                 GPIO0
#define BUMPER3                                 GPIO4
#define BUMPER_FRONT_LEFT                               GPIO_Pin_2 //5.2
#define BUMPER_FRONT_RIGHT                              GPIO_Pin_0 //4.0
#define BUMPER_BACK_LEFT                                GPIO_Pin_7 //1.7
#define BUMPER_BACK_RIGHT                               GPIO_Pin_1 //4.1
#define BUMPER_RIGHT                                    GPIO_Pin_3 //5.5
#define BUMPER_LEFT                                             GPIO_Pin_0 //1.0
#define LEFT                                    1
#define RIGHT                                   0

#define CLOSE                                   8
#define NORMAL                                  4
#define FAR                                         2

#endif

/****************** (C) COPYRIGHT 2007 STMicroelectronics *****END OF FILE****/
```

B.2 Color Histogram Calculator

————color_calculator.m————

```
%% data structure to keep images
R = [];
G = [];
B = [];
%% the sample images should be read 1-by-1
im = imread('table3.png');
fprintf('done \t');


%% the part containing the object should be selected
[sub] = imcrop(im(:,:,1:3));
imshow(sub);
subR = double(sub(:,:,1));
subG = double(sub(:,:,2));
subB = double(sub(:,:,3));
```

```
R = [R subR(:)'];
G = [G subG(:)'];
B = [B subB(:)'];

%save
%load matlab

%% After taking all samples, the histogram drawn automatically

R = R / 16;
G = G / 16;
B = B / 16;

    subplot(1,3,1);
hist(double(R),16)
title('R');
hold on;
    subplot(1,3,2);
hist(double(G),16)
title('G');
hold on;
    subplot(1,3,3);
hist(double(B),16)
title('B');
hold on;

%% bound finding & the mask and the object shown to see the result of
%%the color coding

im = imread('table3.png');

%the bounds for each color channel
%the values that should be pass the microprocessor for color coding
bR = prctile(R,[25-12.5 75+12.5])
bG = prctile(G,[10 90])
bB = prctile(B,[25 75])

maskR(:,:) = im(:,:,1)/16 > floor(bR(1)) & im(:,:,1)/16 < ceil(bR(2));
```

```
maskG(:,:) = im(:,:,2)/16 > floor(bG(1)) & im(:,:,2)/16 < ceil(bG(2));

maskB(:,:) = im(:,:,3)/16 > floor(bB(1)) & im(:,:,3)/16 < ceil(bB(2));

mask =maskB & maskR & maskG;


    subplot(1,3,1);
    hold on
    title('mask R');
    hold on
    imshow(maskR);
    subplot(1,3,2);
    hold on
    title('mask G');
    imshow(maskG);
    subplot(1,3,3);
    hold on
    title('mask B');
    imshow(maskB);
    figure;
    subplot(1,2,1);
    hold on
    title('mask RGB');
    imshow(mask);
    subplot(1,2,2);
    hold on
    title('orginal picture');
    imshow(im);
```

B.3 Object's Position Calculator

----------position_calculator.m----------

```
%% 1-) Ball Position Calculator
% This script does transformation for the 3D ball position to 2D robot
```

```
% Translation vector:
% position of chesboard in camera reference system
Tc_ext = [ 17.929188      -0.849442       64.715079 ];
% Rotation matrix:
Rc_ext = [ -0.042070      -0.998317       -0.039914
    -0.614786     0.057357         -0.786605
    0.787571      -0.008554        -0.616165 ];

%transformation of camera to chess board
T_cch=[Rc_ext Tc_ext'];

H0=[eye(3) [0 0 0]'];

% the position of the origin
image_point = [81 62 1]';

%K-calibration matrix
K=[191.71462 0   80.35911
    0   191.27299 61.27650
    0    0        1        ];
% T_rch indicates the transformation robot to chess board
% x,y,z the distance of the chess board from the robots center in mm.
x=600;
y=0;
z=0;

T_rch =[1    0    0    x
    0    1    0    y
    0    0    1    z    ];
%transformation are rewritten for clarification
% camera to world
T_cw =[T_cch;0 0 0 1];
%world to camera
T_wc = inv(T_cw);
%world to robot
T_wr =[T_rch;0 0 0 1];
% robot to world
T_rw = inv(T_wr);
```

```
position=inv(K)*image_point;
l=sqrt(x^2+y^2);
%l the distance of the origin point selected in the image to the robot
%% center
Dreal=200; %diameter of the ball in mm
Dpx=29;    %pixel counter of diameter in image

%the f* constant pixel/unit (mm)
fstar = Dpx * l / Dreal;
%the position should be normalized since position(3) is not 1
position= position/norm(position);

Pc=[(fstar*Dreal/Dpx)*position;1];

inv(T_cw*T_rw)*Pc;

result= T_wr*T_wc*Pc;

result

%ans gives the position of the ball in real world in mm
% ans(1) x;ans(2) y;ans(3) z; coordinates in real world

%% 2-) Calibration for the object at Ground

%tranlastion matrix-coming from image toolbox
Tc_ext = [ 160.700069     27.498986        492.532634 ];

%rotation matrix-coming from image toolbox
Rc_ext = [ -0.009783      -0.999883        -0.011800
    -0.428525     0.014854         -0.903408
    0.903477      -0.003781        -0.428620 ];
%transformation of camera to chess board
T_cch=[Rc_ext Tc_ext'];
% K internal camera parameters-from image toolbox
K=[191.71462 0   80.35911
    0   191.27299 61.27650
    0    0        1        ];
% x,y,z the distance of the chess board from the robots center in mm.
x=550;
```

```
y=-150;
z=0;
% T_rch indicates the transformation robot to chess board
T_rch =[1    0    0    x
    0    1    0    y
    0    0    1    z    ];
%transformation are rewritten for clarification
% camera to world
T_cch1 =[T_cch;0 0 0 1];
%world to camera
T_chc = inv(T_cch1);
%world to robot
T_rch1 =[T_rch;0 0 0 1];
% robot to world
T_chr = inv(T_rch1);


T_rc=T_rch1*T_chc;
T_cr = inv(T_rc);
T_cr = T_cr(1:3,:);

ROBOT(1,:,:,:) =T_cr(1,:,:,:);
ROBOT(2,:,:,:) =T_cr(2,:,:,:);
ROBOT(3,:,:,:) =T_cr(3,:,:,:);

t_=[1 0 0
    0 1 0
    0 0 0
    0 0 1];
H= K*T_cr*t_;
Hinv=inv(H);


point = Hinv*[72 119 1]';
result= point/point(3);
result
%% 3-) Calibration for servo_top position at ground

%tranlastion m
Tc_ext = [ 100.613176     95.664945        539.117313 ];
```

```
%rotation matrix
Rc_ext = [ 0.025851        -0.998468        0.048927
    -0.364120      -0.054986         -0.929727
    0.930993       0.006219          -0.364984 ];
%transformation of camera to chess board
T_cch=[Rc_ext Tc_ext'];
% K internal camera parameters-from image toolbox
K=[191.71462 0   80.35911
    0   191.27299 61.27650
    0     0         1          ];
% x,y,z the distance of the chess board from the robots center in mm.
x=600;
y=-100;
z=0;
% T_rch indicates the transformation robot to chess board
T_rch =[1    0    0    x
    0    1    0    y
    0    0    1    z    ];
%transformation are rewritten for clarification
% camera to world
T_cch1 = [T_cch;0 0 0 1];
%world to camera
T_chc = inv(T_cch1);
%world to robot
T_rch1 = [T_rch;0 0 0 1];
% robot to world
T_chr = inv(T_rch1);


T_rc = T_rch1*T_chc;
T_cr = inv(T_rc);
T_cr = T_cr(1:3,:);

ROBOT(1,:,:,:) =T_cr(1,:,:,:);
ROBOT(2,:,:,:) =T_cr(2,:,:,:);
ROBOT(3,:,:,:) =T_cr(3,:,:,:);

t_=[1 0 0
    0 1 0
```

```
    0 0 0
    0 0 1];
H= K*T_cr*t_;
Hinv=inv(H);

point = Hinv* [19 99 1]';
result = point/point(3);
result
```

B.4 Motion Simulator

----------robot_distance.m----------

```
%% The distance should be expressed in terms of frontal-lateral
%% (cartesian coordinate, angle to the object (polar coordinate)

X = 3500;
Y= -350;
deltaX=X;
deltaY=Y;
p = sqrt(deltaX*deltaX+deltaY*deltaY);
angle = atan2(Y,X);

alpha = 0;
frontal_speed = 3500;
lateral_speed = -350;

vf= frontal_speed*cos(-alpha)-lateral_speed*sin(-alpha) % new x
vl=frontal_speed*sin(-alpha)+lateral_speed*cos(-alpha)
% new y
%% The triskar function called like this
[vt,omega_w,vn] = TriskarOdometry(250, 24.6, pi/6,vf,vl,angle,1);
```

----------Triskar.m----------

```
function [vt, om, vn] = TriskarOdometry(R_robot, R_wheel, alpha, v_F,
v_L, omega, plot)

% - R_robot: robot radius (by mean of distance of the wheels from the
% center of the robot) [mm]
% 25 cm in our case -- 250 mm
% - R_wheel: wheel radius [mm]
%  24.6000 mm yaricap
% - alpha: angle of the front wheels [rads] (in our case alpha = pi/6)
% - vF: frontal speed setpoint [mm/s]    ---x axis
% - vL: lateral speed setpoint [mm/s], > 0 if oriented to right -y axis
% - omega: angular velocity setpoint [rad/s] > 0 if CCW
% - plot: draw a plot with all vectors (1 = plot, 0 = don't plot)
%
% this function returns:
% - vt: vector of tangential wheel velocities [mm/s]
% - vn: vector of "sliding" velocity of the wheels due to rollers [mm/s],
% > 0 if directed from the center to the extern
% - omega_w: angular velocity of the wheels [mm/s], > 0 if CW looking at
% the wheel from the center of the robot
%
% You can call that function like:
% [vt,omega_w,vn] = TriskarOdometry(250, 24.6, pi/6,1000,1000,0,1)

cosA = cos(alpha);
sinA = sin(alpha);

v = [v_F, v_L, omega*R_robot]'

MF = [-cosA  sinA  -1;
       cosA  sinA  -1;
       0          -1          -1]

ML = [sinA   cosA  0;
      sinA  -cosA  0;
      -1          0          0];

vt = MF*v;
om = vt/R_wheel;
vn = ML*v;
```

```
if(plot == 1)
    % scalature
    k1 = max(abs(v_F),abs(v_L));
    v1 = v_F/k1*R_robot/2;

    v2 = v_L/k1*R_robot/2;

    p1 = R_robot/2;

    m_arr = [0  0.04*p1  -0.04*p1;  0.1*p1  0  0];
    v_txt    = [0.05*p1;  0.2*p1];



    s1 = R_robot/( 2 * max(abs([vt;vn])));

    %plot
     figure(1)

    hold on
    h1 = line([0,R_robot*cosA],[0  R_robot*sinA]);
    h2 = line([0,-R_robot*cosA],[0  R_robot*sinA]);
    h3 = line([0,0],[0  -R_robot]);
    set(h1,'Color',[0.2  0.2  0.2]);
    set(h2,'Color',[0.2  0.2  0.2]);
    set(h3,'Color',[0.2  0.2  0.2]);
    set(h1,'LineStyle','-.');
    set(h2,'LineStyle','-.');
    set(h3,'LineStyle','-.');

    if(v_F ~= 0)
        line([0  0],[0  v1],'Color',[1  0  0]);
        if(v_F < 0)
            fill([0  p1*0.05  -p1*0.05],[v1  v1+0.12*p1  v1+0.12*p1],'r')
        else
            fill([0  p1*0.05  -p1*0.05],[v1  v1-0.12*p1  v1-0.12*p1],'r')
        end
        text(0.15*v1,v1,'v_F');
    end
```

```
    if(v_L ~= 0)
        line([0 v2],[0 0],'Color',[1 0 0]);
        if(v_L < 0)
            fill([v2 v2+0.12*p1 v2+0.12*p1],[0 p1*0.05 -p1*0.05],'r')
        else
            fill([v2 v2-0.12*p1 v2-0.12*p1],[0 p1*0.05 -p1*0.05],'r')
        end

        text(v2,0.1*v2,'v_L');
    end
    if(omega ~= 0)
        if(omega > 0)
            theta = linspace(-pi/3,pi/2,100);
            fill([-0.12*p1 0 0],[R_robot/6 R_robot/6+p1*0.05 R_robot/6-p1
*0.05],'b');
            text(-0.14*p1,R_robot/6+p1*0.05,'\omega')
        else
            theta = linspace(pi/2,4*pi/3,100);
            fill([0.12*p1 0 0],[R_robot/6 R_robot/6+p1*0.05 R_robot/6-p1*
0.05],'b');
            text(0.12*p1,R_robot/6+p1*0.05,'\omega');
        end
        rho = ones(1,100)*R_robot/6;
        [xr,yr] = pol2cart(theta,rho);
        line(xr,yr);
    end
    % ruota 1

    if(vt(1) ~= 0)
    line([R_robot*cosA, R_robot*cosA+vt(1)*s1*sinA],[R_robot*sinA, R_robo
*sinA-vt(1)*s1*cosA],'Color',[0 1 0]);
    offset1 = [R_robot*cosA+vt(1)*s1*sinA; R_robot*sinA-vt(1)*s1*cosA];
    if(vt(1)*s1*sinA < 0)
        M = m_rot(alpha);
    else
        M = m_rot(-pi + alpha);
    end

    m_arr1 = M*m_arr + [offset1 offset1 offset1];
    v_txt1 = M*v_txt + offset1;
```

```
    fill(m_arr1(1,:),m_arr1(2,:),'g');
    text(v_txt1(1),v_txt1(2),'v_{t1}');
    end

    if(vn(1) ~= 0)
    line([R_robot*cosA, R_robot*cosA+vn(1)*s1*cosA],[R_robot*sinA, R_robo
sinA+vn(1)*s1*sinA],'Color',[0 1 0])
    offset1 = [R_robot*cosA+vn(1)*s1*cosA; R_robot*sinA+vn(1)*s1*sinA];
    if(vn(1)*s1*cosA < 0)
        M = m_rot(pi/2+alpha);
    else
        M = m_rot(-pi/2 + alpha);
    end

    m_arr1 = M*m_arr + [offset1 offset1 offset1];
    v_txt1 = M*v_txt + offset1;
    fill(m_arr1(1,:),m_arr1(2,:),'g');
    text(v_txt1(1),v_txt1(2),'v_{n1}');
    end

    % ruota 2
    if(vt(2) ~= 0)
    line([-R_robot*cosA, -R_robot*cosA+vt(2)*s1*sinA],[R_robot*sinA, R_ro
*sinA+vt(2)*s1*cosA],'Color',[0 1 0]);
    offset1 = [-R_robot*cosA+vt(2)*s1*sinA; R_robot*sinA+vt(2)*s1*cosA];
    if(vt(2)*s1*sinA < 0)
        M = m_rot(-pi-alpha);
    else
        M = m_rot(-alpha);
    end

    m_arr1 = M*m_arr + [offset1 offset1 offset1];
    v_txt1 = M*v_txt + offset1;
    fill(m_arr1(1,:),m_arr1(2,:),'g');
    text(v_txt1(1),v_txt1(2),'v_{t2}');
    end

    if(vn(2) ~= 0)
    line([-R_robot*cosA, -R_robot*cosA-vn(2)*s1*cosA],[R_robot*sinA, R_ro
*sinA+vn(2)*s1*sinA],'Color',[0 1 0])
```

```
offset1 = [-R_robot*cosA-vn(2)*s1*cosA; R_robot*sinA+vn(2)*s1*sinA];
if(vn(2)*s1*sinA < 0)
    M = m_rot(-pi/2-alpha);
else
    M = m_rot(pi/2-alpha);
end

m_arr1 = M*m_arr + [offset1 offset1 offset1];
v_txt1 = M*v_txt + offset1;
fill(m_arr1(1,:),m_arr1(2,:),'g');
text(v_txt1(1),v_txt1(2),'v_{n2}');
end

% ruota 3
if(vt(3) ~= 0)
line([0, -vt(3)*s1],[-R_robot, -R_robot],'Color',[0 1 0]);
offset1 = [-vt(3)*s1; -R_robot];
if(-vt(3)*s1 < 0)
    M = m_rot(pi/2);
else
    M = m_rot(-pi/2);
end

m_arr1 = M*m_arr + [offset1 offset1 offset1];
v_txt1 = M*v_txt + offset1;
fill(m_arr1(1,:),m_arr1(2,:),'g');
text(v_txt1(1),v_txt1(2),'v_{t3}');
end

if(vn(3) ~= 0)
line([0, 0],[-R_robot, -R_robot-vn(3)*s1],'Color',[0 1 0])
offset1 = [0; -R_robot-vn(3)*s1];
if(-vn(3)*s1 < 0)
    M = m_rot(pi);
else
    M = m_rot(0);
end
m_arr1 = M*m_arr + [offset1 offset1 offset1];
v_txt1 = M*v_txt + offset1;
fill(m_arr1(1,:),m_arr1(2,:),'g');
```

```
    text(v_txt1(1),v_txt1(2),'v_{n3}');
    end

    y_ref = 0;
    delta = -0.095*R_robot;
    text(R_robot*0.6,y_ref,['v_{t1}: ',num2str(vt(1)),' mm/s'])
    text(-R_robot*0.8,y_ref,['v_{F}: ',num2str(v_F),' mm/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['v_{n1}: ',num2str(vn(1)),' mm/s'])
    text(-R_robot*0.8,y_ref,['v_{L}: ',num2str(v_L),' mm/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['v_{t2}: ',num2str(vt(2)),' mm/s'])
    text(-R_robot*0.8,y_ref,['\omega_{R}: ',num2str(omega),' rad/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['v_{n2}: ',num2str(vn(2)),' mm/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['v_{t3}: ',num2str(vt(3)),' mm/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['v_{n3}: ',num2str(vn(3)),' mm/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['\omega_1: ',num2str(om(1)),' rad/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['\omega_2: ',num2str(om(2)),' rad/s'])
    y_ref = y_ref + delta;
    text(R_robot*0.6,y_ref,['\omega_3: ',num2str(om(3)),' rad/s'])

    axis equal
    hold off
end
```

Appendix C

User Manual

This document gives information about the robot, the settings that needs to be made before the starting a run. We use two different programs in the robot. The first one is the ST software, that is used during the color selection and other demo features provided with the software. The second one is the software which we implemented for the Thesis, the game software. In order to run the game, we need also the ST color selection interface, since it offers a good visual feedback on the selection process. The ST program uses the QT Framework, and installation is necessary to use the ST color selection interface.

C.1 Tool-chain Software

In order to implement the robot, to flash the written software in to the microprocessor, we use the ST-ARM Toolchain. Unzip the archive "STLCam XXXz Redistrib.zip" in the Eclipse workspace directory. Now you need to import the project. From Eclipse choose File-Import-General-Existing project into workspace, browse for the "STLCam" directory and then click "Finish" shown in Figure C.1.

After the import completed, and the compilation ended successfully, we need to import the "Launch Configurations" to the tool-chain. From Eclipse choose File-Import-Run/Debug-Launch Configurations, and browse for the "Launch Configurations 003" and then click "Finish". The step can be seen from the Figures C.2, C.3 below.

Before programming the microprocessor, the drivers for the "JTAG pro-

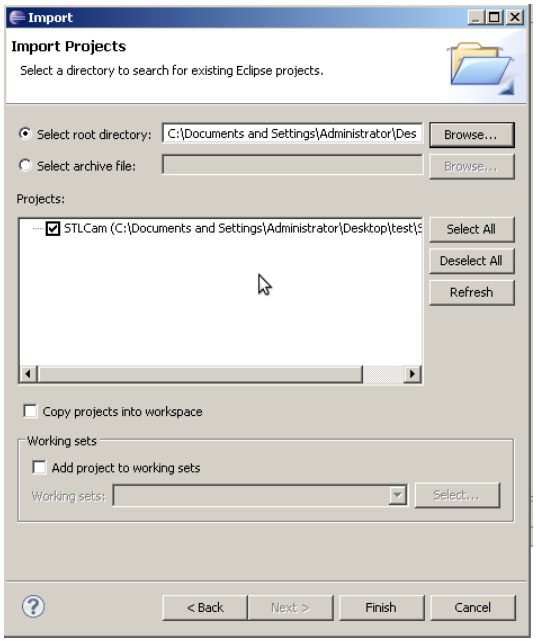

Figure C.1: The import screen of Eclipse

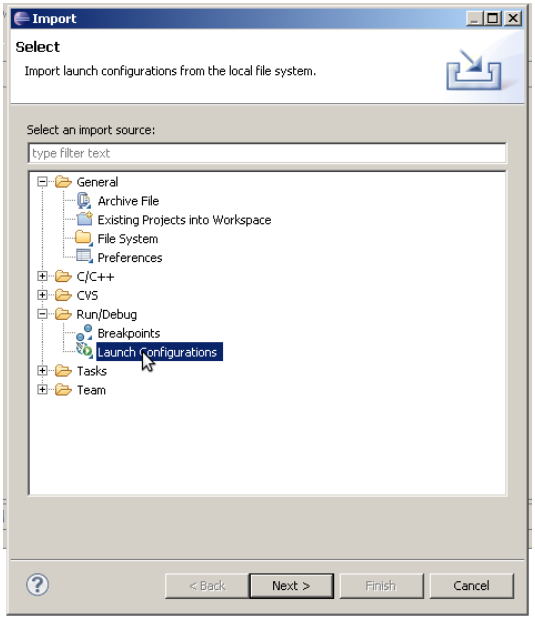

Figure C.2: The import screen for Launch Configurations

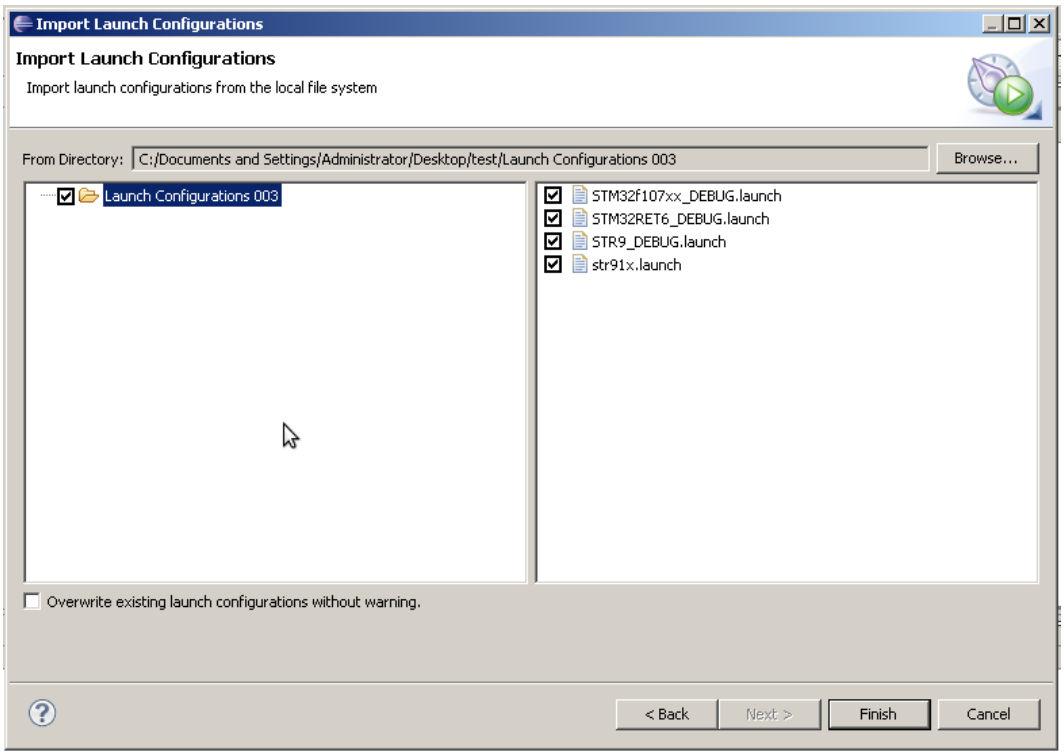

Figure C.3: The second step at importing Launch Configurations

grammmer“ and the ”Virtual COM Port“ for the USB driver of the board, provided in the CD, should be installed. The PC should be restarted if it is necessary.

After installing the drivers successfully, the next step is flashing ST's software into the microprocessor. You can program the microprocessor by clicking the ”program_USB“ from the ”Make“ view.

If the programming is finished successfully, you should see ”Flash Programming Finished.“ from the ”Console“.

C.2 Setting up the environment (Qt SDK Open-source)

Download the 2009.04 version of the Qt SDK from here: ftp://ftp.qt.nokia.com/qtsdk/qt-sdk-win-opensource-2009.04.exe Do not download the last version, at the moment our software works only with Qt 4.5.x. Run the executable that will install on your pc the Qt Framework 4.5.3 and the Qt Creator IDE.

Unzip the archive "STRVS-Serial Executable.zip" in a directory of your choice. If you receive the following error (Figure C.4) while trying to launch the "STRVS-Serial 046.exe", place the executable file in the installed directory of the Qt Framework and launch from that location.

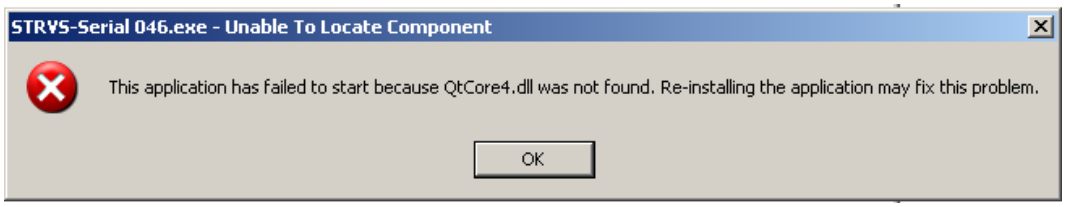

Figure C.4: The error that can be caused if Qt libraries not found. To solve the problem, place the executable file under Qt libraries folder.

From the "Windows Device Manager" under "Ports(COM & LPT)" find the "STM Virtual COM Port" and check port number. In the example it is "COM4" (Figure C.5). Use this information in the software. In the following order, we first "Open" the port. Then "Start Camera" with the "BlobSearch" option. Then "Get Image" to check the camera started correctly. "Continuous" should be selected if we can to see the image as a video stream (Figure C.6).

Since there is no mechanism to save the pictures, we capture the pictures by taking a "snapshot" in the windows. The important point in capturing the images is to set "Zoom" parameters to "%100" in order not to change the resolution of 160x120. The images should be cut also at the orginal resolution (160x120) from the "snapshot" images to achieve the correct values. The matlab script "color_calculator" that can be found at Appendix B.1, is used to calculate the color histogram of the demanded object. It is advised to take as many samples as you can, with different distances to robot and with different illumination to have better histogram.

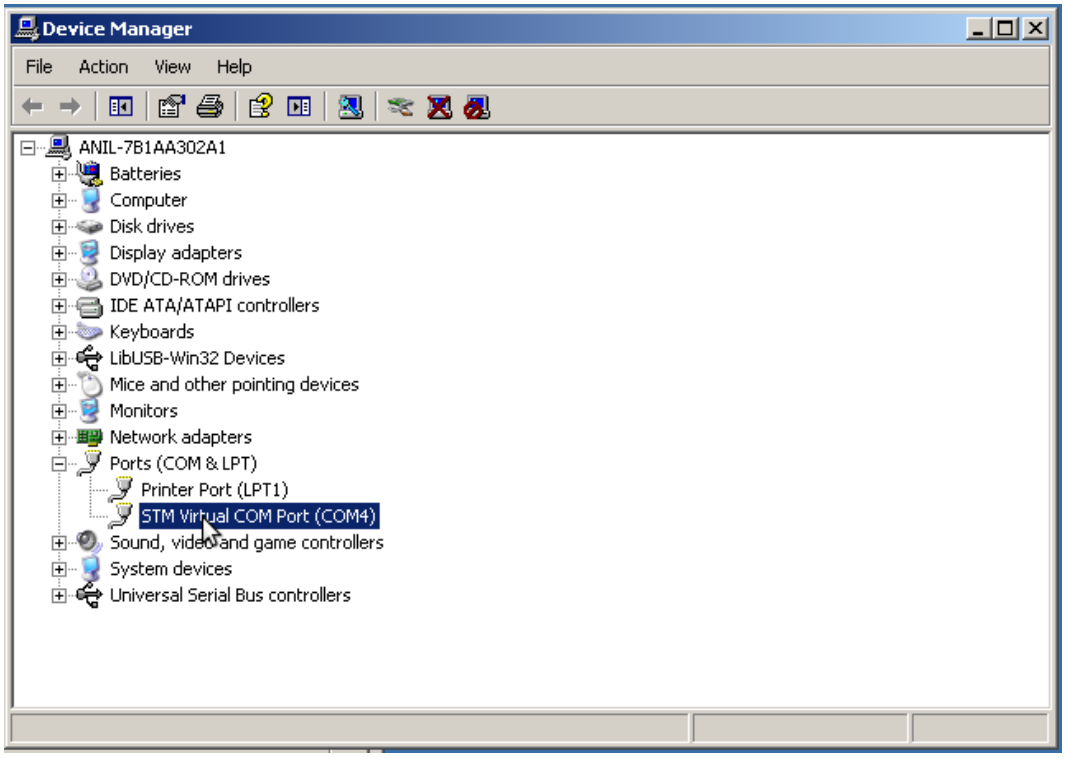

Figure C.5: The drivers can be validated from Device Manager

To test the color found with "color_calculator" the following steps should be followed. From the tabs at the top, "BlobSearch" should be selected. Inside "BlobSearch" tab, color class number should be set accordingly with the number of the color we want to control (Figure C.7). Under the "Blob Color" tab, "Visual Rule Editor" should be checked to control the color. The color code defined with inside the "main software" for the thesis (will be explained in details in a few steps) could be tested at that tab, by adding the color codes to the "Rules List" that can be found on the lower right bottom of the software.

When the color information is ready, it should be captured by clicking the "Get from Visual Rule Editor" in the "Custom" menu. The next step is setting the "Blob Geometry" that defines the size of the color we are searching for. After finishing these steps, we should deselecting the "Visual Rule Editor" and "Save settings". In order to test the whole process, we should select "continuous" and click "Get Blobs". The selected color will be

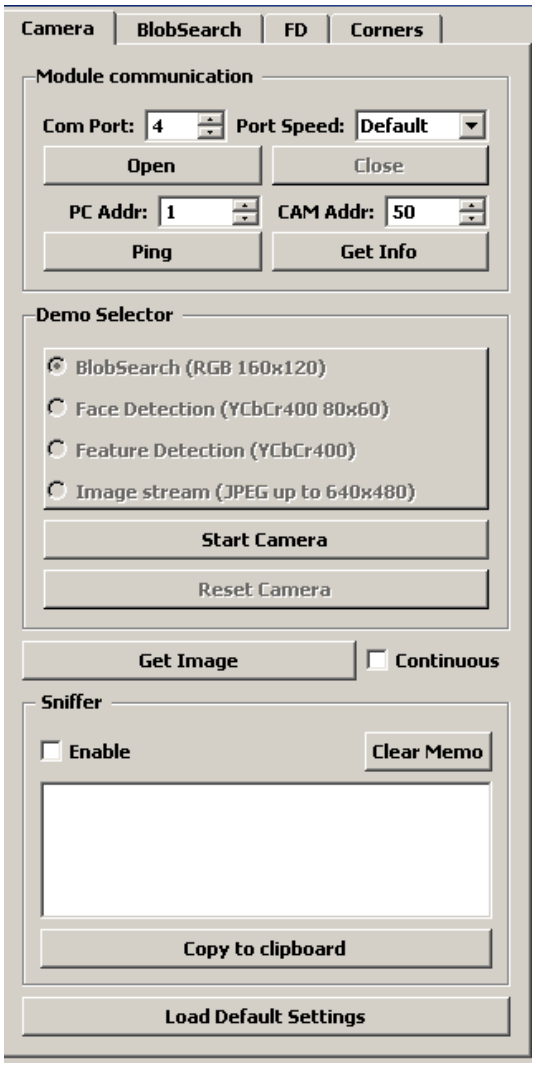

Figure C.6: The main screen of the ST's software

represented according to the parameters defined under the "Visualization" tabs (Figure C.7).

C.3 Main Software - Game software

The project should be imported to the Eclipse in the same way with the ST's software. The color information and the blob geometry parameters are found under the source-Camera_Init.c . The function that contains these

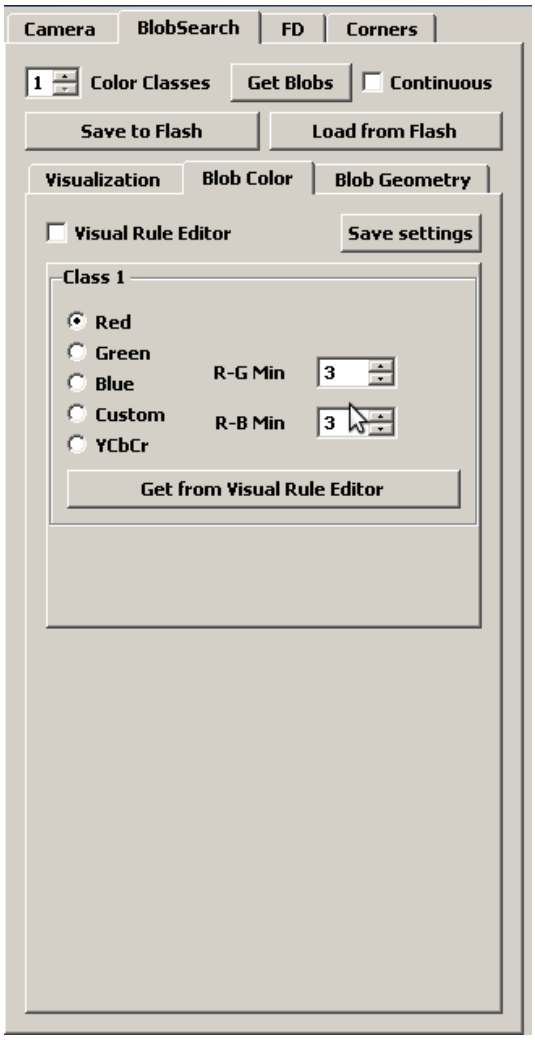

Figure C.7: The second step at importing Launch Configurations

information are placed in the Init_BlobParameters function. BlobSearchSetColParam(u8 IDCol, u16 MinPer, u16 MaxPer, u16 MinArea, u16 MaxArea, u8 MinCirc, u8 MaxCirc) taking the parameters defined, should be updated with the values found in the previous step, with ST's software. "IDCol" is the color class defined and it is be the same also with the corresponding RGBCube_AddCube(u8 RMin, u8 RMax, u8 GMin, u8 GMax, u8 BMin, u8 BMax, u8 IDCol); and RGBCube_SubCube(u8 RMin, u8 RMax, u8 GMin, u8 GMax, u8 BMin, u8 BMax, u8 IDCol); functions that defines the color coding. The values that are tested with ST's software should be updated in RGBCube_AddCube and RGBCube_SubCube functions.

The robot is ready to play the game, after building the code and programming the microprocessor through "program_USB" that can be found in Eclipse from the "Make" view (Figure C.8).

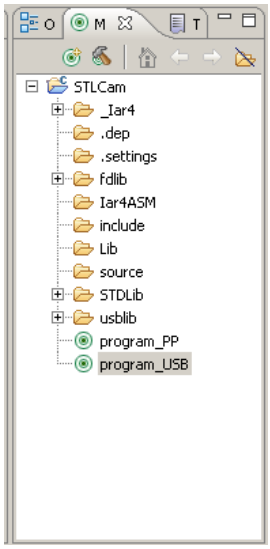

Figure C.8: The programming of the microprocessor is done through "program_USB. The process can be controlled through the console.

Appendix D

Datasheet

The pin mappings of the board to the corresponding GPIO port of the microcontroller can be seen in the figure D.3.

Mainly GPIO pins 1.6, 1.4, 1.3 represents the PWM wave outputs for the motor1, motor2, motor3 in the same order. The pins 0.6, 0.2, 0.4 and 0.7, 0.3, 0.5 are the direction pins for the motor control; that is used to control which direction is forward, which is reverse and also to break-down the motors. The pins 5.3, 5,2, 1.0, 4.0, 4.1 are for the tactile bumper sensors. The detailed description can be found in "definitions.h" inside main software.

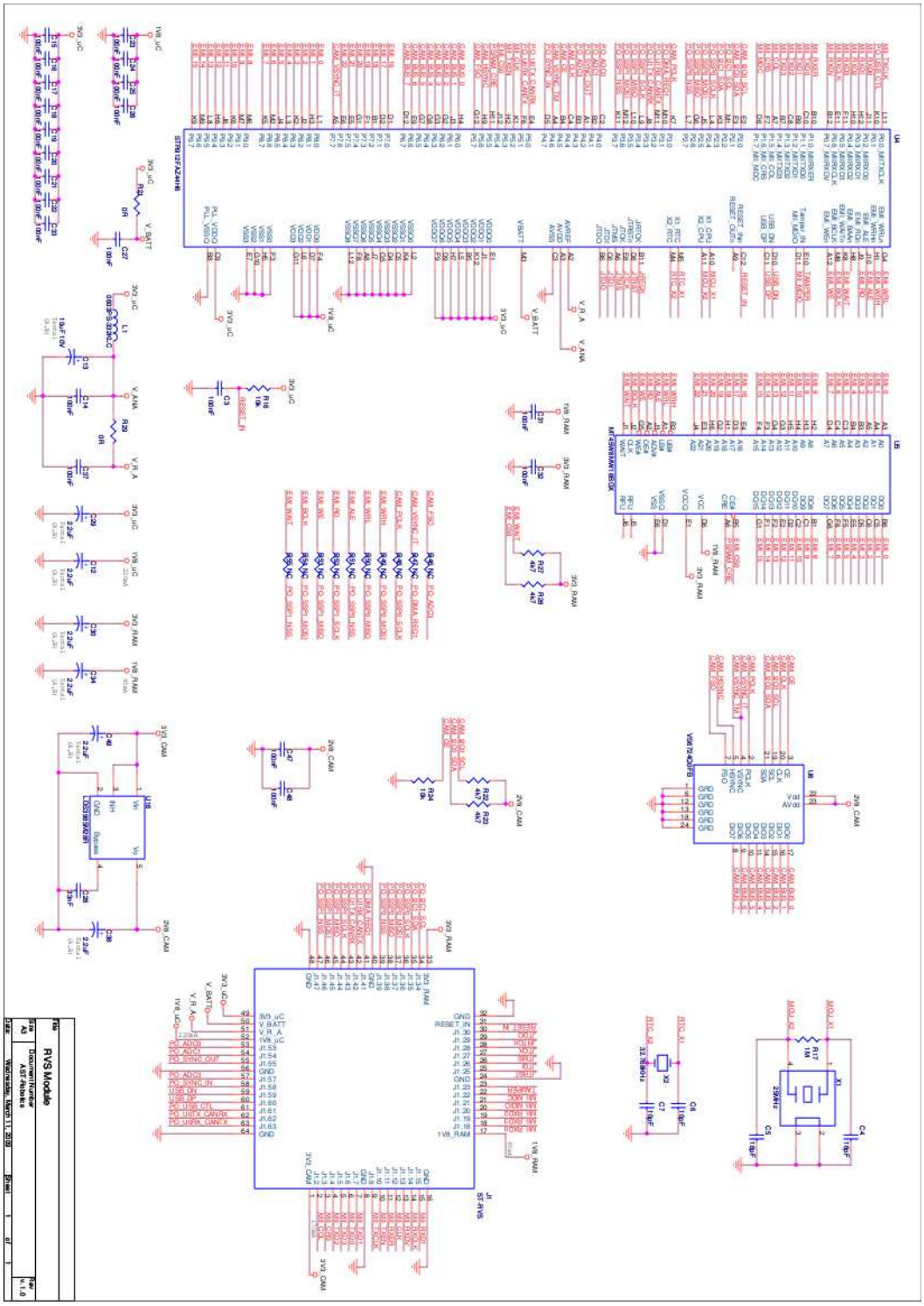

Figure D.1: The schematics for the RVS Module board

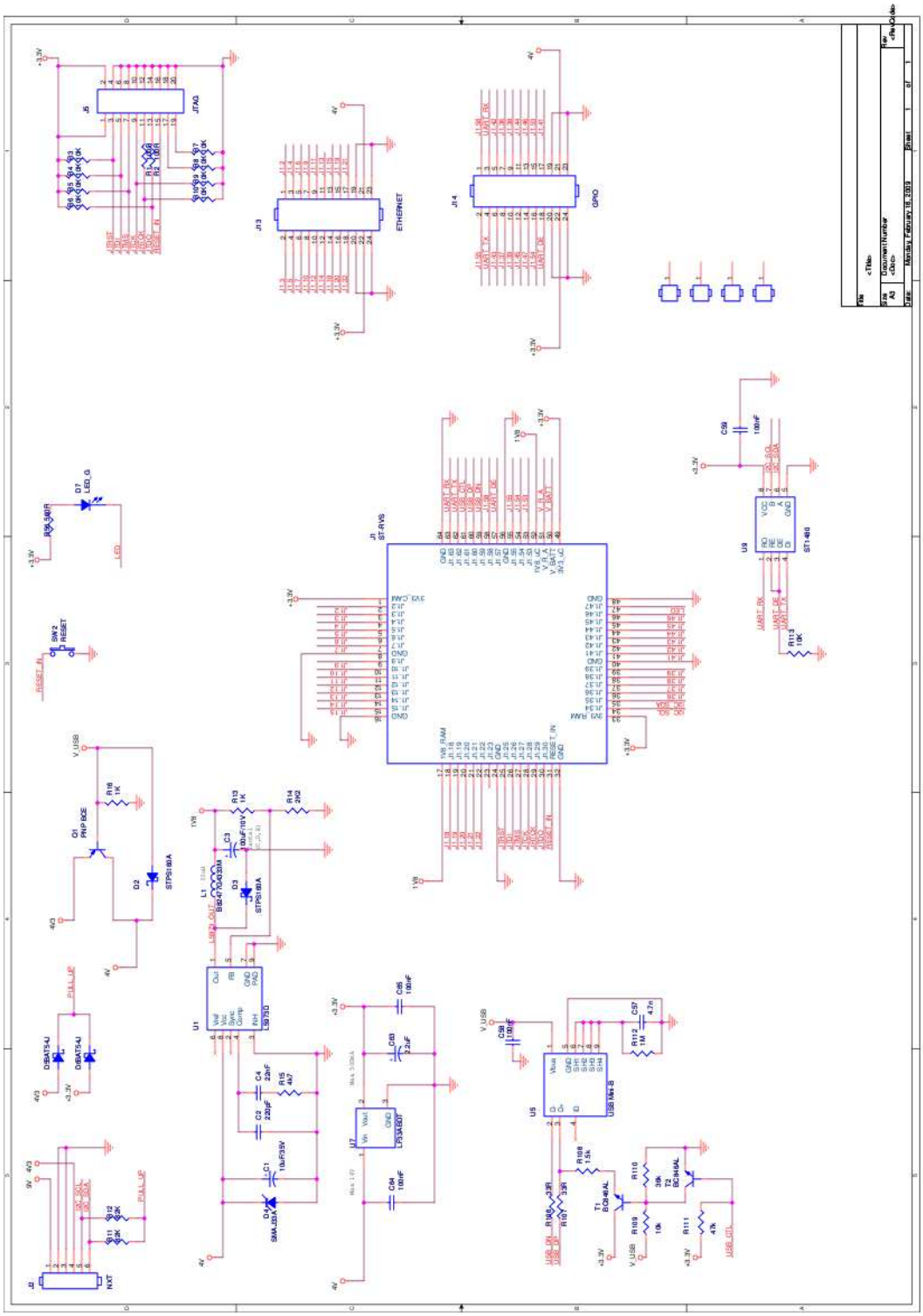

Figure D.2: The schematics for the STL Mainboard

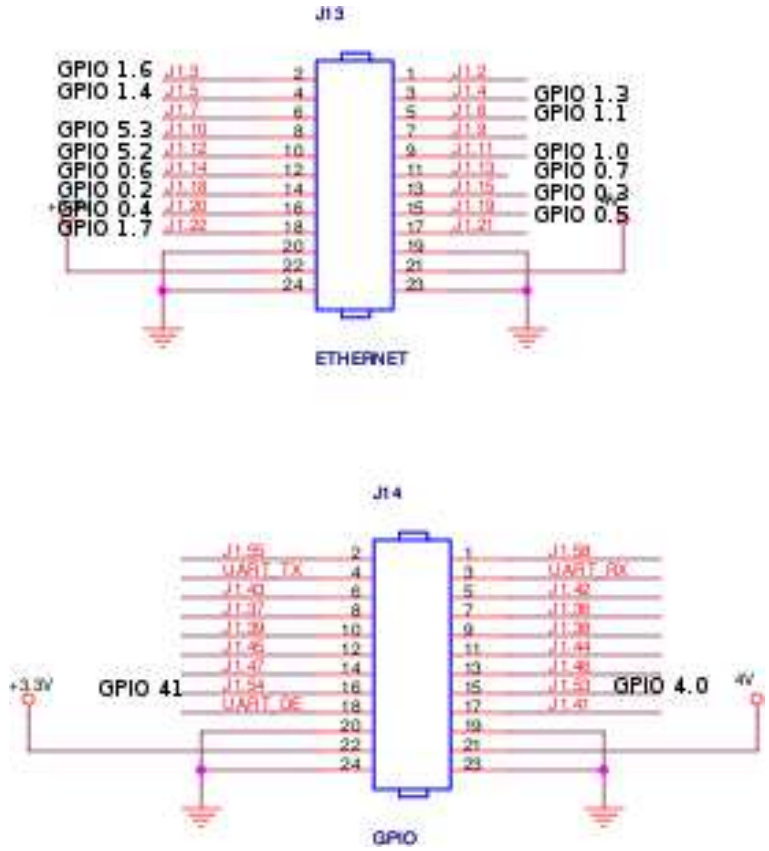

Figure D.3: The pin-mappings of the board with corresponding GPIO in the microcontroller