Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Fraunhofer Institut Produktionstechnik und Automatisierung

PROJECTE DE FI DE CARRERA

TÍTOL DEL PFC: Demostrador Robòtic de Mapejat d'Interiors 3D per l'Institut Fraunhofer

TITULACIÓ: Enginyeria de Telecomunicació (segon cicle)

AUTOR: Julio Sagardoy Pérez

DIRECTOR: Sílvia Ruiz

SUPERVISOR: Dipl.Ing. Georg Arbeiter

DATA: 15 de març de 2012

I would like to thank all these people who, in whatever the way, have helped and supported me during these 6 months: first of all I want to specially thank Georg Arbeiter, for giving me this great opportunity of incalculable value; Florian Weißhardt and Nadia Hammoudeh for the tech support on ROS; Matthias from electronics for his unconditional support with the mechanical and electronics stuff. Txus Melo and Mercè Agüera from the International Relationships Office; Mom and Dad and my brother; my friends here and in Barcelona, the guys at the student room, and all those people that are not in this list... but that were just there.

Títol: Demostrador Robòtic de Mapejat d'Interiors 3D per l'Institut Fraunhofer

Autor: Julio Sagardoy Pérez

Director: Sílvia Ruiz

Data: 15 de març de 2012

Resum

Dins del món de la robòtica, la branca de la visió computeritzada ha estat desde sempre un dels aspectes més curiosos i complexes. Per definició, la visió computeritzada comprèn tots aquells mètodes per obtenir, processar i transformar l'entorn del robot en una representació numèrica, que pugui ser entesa per la unitat central de processament del robot. D'aquesta manera, el robot pot conèixer la seva posició i pot computar moviments, accions i decisions depenent d'aquest entorn.

Els investigadors de l'institut Fraunhofer IPA estàn actualment desenvolupant la tercera generació del seu exitós robot d'assistència personal Care-O-Bot®. Un dels aspectes que més impresiona d'aquest robot és la capacitat per moure's per l'entorn, evitant i esquivant obstacles, agafant objectes i canviant-los de posició, però sobretot, la capacitat d'interaccionar amb humans. Sota el capó, el que trobem es una enorme quantitat de sensors, que permeten al Care-O-Bot® 3 de detectar l'entorn en què opera.

El Care-O-Bot® 3 també és utilitzat com a plataforma de recerca pel mateix institut. El projecte en questió neix de la necessitat de tenir una plataforma de recerca i desenvolupament més senzilla i fàcil d'utilitzar que el Care-O-Bot® 3, per tal de ser utilitzada com a demostrador de les tecnologies de mapejat esmentades. Així doncs, l'objectiu d'aquest projecte ha estat la concepció, el desenvolupament i la construcció d'un demostrador robòtic de mapejat, de manera que pugui ser fet servir pels investigadors de l'insitut Fraunhofer IPA.

Title: 3D Mapping Demonstrator for Fraunhofer IPA

Author: Julio Sagardoy Pérez

Director: Sílvia Ruiz

Date: March, 15th 2012

Overview

Within the robotics universe, the computer vision branch is one of the most curious and complex aspects. Per definition, computer vision field comprises the methods for acquiring, processing and transforming the environment of the robot to numerical methods, so its central processing unit can understand its location and compute movements, actions and decisions depending on this surrounding world.

Researchers at Fraunhofer IPA are currently developing the third generation of their successful Care-O-Bot® home assistance robot. One of the aspects that impresses more about this robot is its capability to move around the environment, avoiding and moving around obstacles, grasping objects and interacting with humans. Under the hood, one can find a multiplicity of sensors enables Care-O-bot® 3 to detect the environment in which it is operating. These range from stereo vision color cameras and laser scanners to a 3D depth-image camera.

The Care-O-Bot® 3 is also used as a research platform within the Fraunhofer IPA. This project was born from the necessity of having a smaller platform to test and demonstrate these computer vision systems. Hence, the objective was the conception, development and construction of a robotic mapping demonstrator platform, so the researchers at the Fraunhofer IPA could use it to demonstrate the advances in computer vision by using an easy to transport, light system.

INDEX

TABLE OF FIGURES

ABBREVIATIONS

ASCII: American Standard Code for Information Interchange

CCW: Counter-Clock-Wise

CLK: Clock

CTC: Clear Timer on Compare match

CW: Clock-Wise

EMC: Electromagnetic Compatibility

EMI: Electromagnetic Interference

GND: Ground

ICSP: In Circuit Serial Programming

LDO: Low Drop-Out

LED: Light Emitting Diode

PCB: Printed Circuit Board

POR: Power On Reset

PWM: Pulse-Width Modulation

RST: Reset

URDF: Unified Robot Description Format

USART: Universal Asynchronous Receiver/Transmitter

USB: Universal Serial Bus

XML: eXtensible Markup Language

1. INTRODUCTION

1.1. Abstract

Personal robotics have always been of interest for the everyday use at home. Several examples and attempts exist since immemorial times, but it is nowadays when presumably practical personal robots are increasingly gaining interest among industry and research. This is mainly because the current state of the art in mechatronics and computer science is now permitting the development of high-tech, incredibly advanced robots which are relatively simpler, reasonably lighter and easier to manufacture and maintain -thus practical- than initial attempts several years ago. The lack of a advanced computer technologies made these cumbersome systems to be mechanical-based, relying mostly on mechanically actuated gears, spurs, joints, and complex synchronism systems to achieve simple pseudo-natural human movements. Today, the availability of powerful computer hardware and software systems and the flexibility these permit, has led to the development of more effective, more practical robots.2

Within the robotics universe, the computer vision branch is one of the most curious and complex aspects. Per definition, computer vision field comprises the methods for acquiring, processing and transforming the environment of the robot to numerical methods, so its central processing unit can understand its location and compute movements, actions and decisions depending on this surrounding world.

Researchers at Fraunhofer IPA are currently developing the third generation of their successful Care-O-Bot® home assistance robot. One of the aspects that impresses more about this robot is its capability to move around the environment, avoiding and moving around obstacles, grasping objects and interacting with humans. Under the hood, one can find a multiplicity of sensors enables Care-O-bot® 3 to detect the environment in which it is operating. These range from stereo vision color cameras and laser scanners to a 3D depth-image camera.

The Care-O-Bot® 3 is also used as a research platform within the Fraunhofer IPA. This project was born

from the necessity of having a smaller platform to test and demonstrate these computer vision systems. Hence, the objective was the conception, development and construction of a robotic mapping demonstrator platform, so the researchers at the Fraunhofer IPA could use it to demonstrate the advances in computer vision by using an easy to transport, light system.

1.2. Project motivation

Getting a place at the Fraunhofer IPA was one of the best opportunities I could have had. Not only because of the widely known reputation of the institution, but also because it served as a start shot, as my intentions since I ended high-school included moving abroad at some point. This decision primarily came because I thought -and still think- that a stay abroad during studies is one of the most enriching experiences one can have in a life. The fact that it is so easy for us -as students- to move abroad for a time also helped: In short, we just have to say where we want to go, sign on a list, and wait for an answer. My case became a bit more complicated than this, but I made it. And I can definitely say that it was worth the effort.

About my stay in the Fraunhofer IPA, I can say I have learnt more than I ever expected. The project spanned by three branches: mechanical engineering, electronics engineering and software/control engineering -the so called mechatronics. And all the three parts had to be done from scratch. This pressed me to gather every single information and the know-how by myself, and then conceive and build the entire working system in less than 5 months, adjusting to the due date.

It has been one of the most enriching things I have done in my life, and an experience I will never forget.

1.3. Document outline

The project documentation is structured following the timeline that the project development followed.

The project started with a mechanical conception of the system, with requirements definition, materials research construction and testing. The first chapter of the work analyzes these requirements, and details in most possible detail the structure construction.

Short after the structure was ready, the electronics for orientating and controlling the platform giving movement to the environment detection sensors had to be developed and tested. After the introductory Chapter 1, Chapter 2 describes how this work was accomplished, providing with all kind of information and electronics details.

Then, a generic protocol for communication the robot was developed, and adapted to the current existing software of the other robots developed by the

Fraunhofer IPA, so it could resemble and use most of their features, but also be easier to use and understand by the researchers once the robot was released to them. Chapter 4 describes this generic protocol, as a reference. Chapter 5 describes the current software structure of the ROS system used at the Fraunhofer IPA, and also describes what was done in order to interact with the robot with it.

2. MECHANICAL STRUCTURE

If one takes a look to the robot itself, it can easily distinguish two separate parts. Actually, these parts can effectively be detached. The first is the robotic, movable platform itself, which will be referred as the “head” from now on. This moving platform contains the sensors in its top, but has also a fixed base, which is attached to the second part. This second part is just a static holding structure which holds the head at a distance of about 150 centimeters above the floor, emulating the Care-O-Bot 3® head location. The base of this structure also acts as a cover for the electronics, as well as storage space for the interconnectivity and power cables.

This first chapter of the project describes in detail how the physical structure of the robot was conceived and designed. Some drawings and schematics of both the head and the holding structure are included in the Appendix.

2.1. Design considerations

Since the early stages of the project, some requirements had to be considered. For example, and most important, the moving platform was required to provide an omnidirectional movement. Also, it had to be possible to attach several accessories -such as a XBOX Kinect device, an ASUS X-Tion device, a camera or other sensors. These requirements regarding the most mechanical part of the project are summarized below:

- Two-degrees of freedom:
    - Pan movement (rotation of the Z axis) of 360º. As room scan patterns are mostly along the horizontal, the Y axis was given a more than 360º rotation permissibility, only limited by the servo cable, which gets twisted around the head at each rotation and thus limits its a continuous, unlimited movement.
    - Tilt movement of 180º. Along with the 360º pan rotation, to complete a full omnidirectional scan, the tilt mechanism was only needed to allow 180º movements, which translate in a tilt-up, tilt-down pattern.
- Closed-loop system: Instantaneous position report of both the pan and tilt mechanisms. The reason for this is twofold: First, the software requires the knowledge of current position at all times to do part of the calculations. Second, a closed loop system will permit more precision and repeatability, and also instantaneous position correction.
- Enough driving torque to drive relatively heavy loads: Although the robot was designed for XBOX Kinect and ASUS X-Tion devices, it may be able

to drive other heavier loads due to its high-torque driving mechanisms. This proved somewhat troublesome to achieve, at least with the selected stepper motor which drives the pan platform, as by design it did not have that much torque at the supplied voltage. Higher loads provoked higher rebounces at each step jump, and depending on the inter-step speed, it could miss jumps. However, it worked perfectly fine for loads similar to the intended devices.

- Heavy-duty construction materials for a reliable operation and robustness during transport, but also for ensuring a long lifespan.

2.2. Pan and tilt mechanism

The so-called head was constructed using steel and aluminum. Its design was done using Dassault Systèmes SolidWorks® 2010 software.

Recall that it is comprised by two movable parts:

- The pan platform, which rotates along the Z axis, with reference to the supporting structure. It is able to rotate indefinitely, as it is acted by a stepper motor. However, the servo wire does not allow turning further than two rotations from initial position. There is no mechanical locking mechanism to avoid this condition, and therefore this is controlled by hardware and software. Because the stepper is not powerful enough to overcome this position, the only effect will be the stepper trying to advance but not being able to.

- The tilt platform, which rotates up to 180º respect the pan platform. The sensors are mounted on top of this platform. This platform is acted by a 180º analog servo.

All the screws in the head are M3 hex-socket-head screws, except for a M4 self-tapping hex screw used for joining the tilt platform to the pan platform.

The cables of both the stepper and the servo are harnessed together. In fact, until almost the end of the project development, the electronics where located in a shell next to the head. However, this was changed, and finally the head cables and those from the sensor head, in this case an ASUS X-Tion, were brought in to the structure to then run down to the base. This makes possible to conveniently hide all the cables, as detailed in subsection

2.3. Supporting structure

The supporting structure is constructed with Bosch-Rexroth profiles. This delivers the great construction, flexibility and robustness provided by the profiles and the relative lower cost when compared to designing a specific, monolithic body. Several profiles of different lengths are mated together forming a base block with a vertical pole, which holds the head. In order to enhance the visual

appearance of this structure, all the visible profile faces are covered with a sleek-looking, aluminum-plastic laminated (Alucobond®) sheets.

Regarding the union between the vertical pole and the head, it was decided to simply make the head directly attachable to the profile with some screws. This is why the lower part of the head describes a square, 45 mm per side shape. Each side has two holes of 4 mm and 5 mm diameter to allow secure and tight fix to the supporting structure. The head gets the vertical profile end inserted in its base, and by means of screws, it stays tightly attached to it. The screws are M5, and are tightened to the profile using Bosch-Rexroth T-Nuts.

Figure 2.1 below shows the profile cross-section:

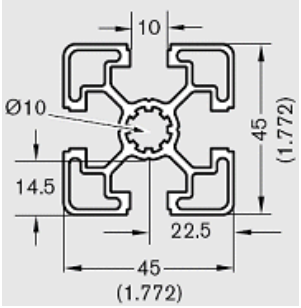

Figure 2.1 Bosch-Rexroth 45 profile cross-section

The support base acts both as pedestal and as enclosure. Two transversal struts hold tightly the vertical pole in place. The rest of the structure follows a prismatic shape, which skeleton is covered with Alucobond® sheets, by means of T-Nuts. This enclosure serves to protect and store the electronics box, as well as the cables.

In the vertical strut pole, the profile central round shape and the T-shaped rails at each face of the profile are used for hiding the cables running from the head to the base. Note that these faces are then covered with removable Alucobond® sheets too, so cables are totally hidden and protected. Alucobond® sheets are therefore attached using equally spaced T-nuts all along the T rails of the vertical pole.

Wheels to allow easy transport of the robot are provided as well. These wheels have a locking mechanism for locking the robot from undesirable movement during operation.

Figure 2.2 below shows the CAD (SolidWorks®) of the structure, without and with the Alucobond® covers. Although not represented in it, the vertical supporting pole is also covered with Alucobond®.

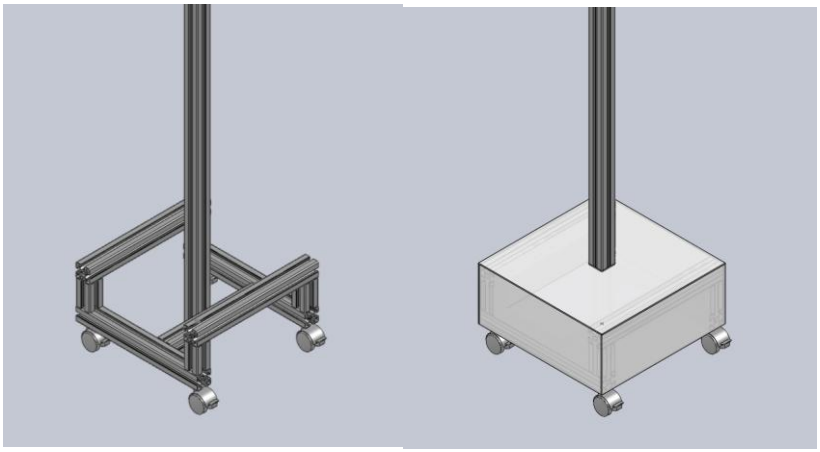

Figure 2.2 Supporting base detail, without and with cover

3. ELECTRONICS

3.1. Introduction

This chapter describes the electronics within the mapping demonstrator robot. The electronics are in charge of interpreting the commands received from the host computer, controlling the movement of the stepper motor and servo, and also reporting the current positions and status of the robot.

The electronics are enclosed within a small plastic box, and are formed by two PCBs (Figure 3.1). The top one is a breadboard containing the microcontroller and a USB to serial converter. This breadboard is attached to another PCB below, which essentially contains the circuitry to control the stepper motor, headers and connectors and a power regulation stage for the entire circuit.

Initially, the electronics box was intended to be a detachable module in the robot head. That would permit easy replacement by another one, in case the electronics stopped working during a demonstration. However, after testing and thorough use, this event proved highly unlikely. This, added to the fact that this “modular look” in the robot head did not look that good, eventually turned down this idea, and the electronics box was placed inside the structure base.

The electronics design was done entirely with Altium® Designer, because the author already had much experience with it and it was available for use at Fraunhofer IPA.

Next Figure 3.1 below shows the overall result. It shows the electronics box without the top cover. Note the microcontroller is located in the PCB on the top, and the lower one shows the head connectors (blue header) and the power connector. When closed, the box measures roughly 80x70x40 mm.

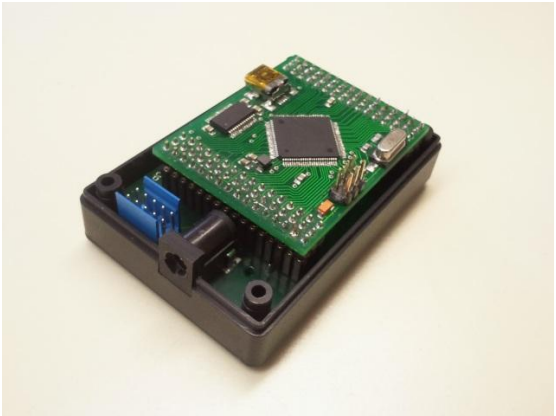

Figure 3.1 Electronics box without cover

This chapter is distributed as follows: First, a brief introduction to the ATmega1280 microcontroller and the peripherals in use. After it, the reader will find a description of how the used stepper controller circuits work. Third, there is a brief explanation on how the servo is controlled. Fourth and finally, how the encoder signals are treated and why for. Electronics schematics and wiring diagrams are included in the Annexes part.

3.2. Breadboard for ATmega1280 microcontroller

The core of the system is based in an AVR ATmega1280 microcontroller. The decision to use such a powerful microcontroller for this application may seem exaggerated, but the fact was that breadboards with the ATmega1280 where available at the Fraunhofer IPA for prototyping. These boards not only have easy access to all the pins through 2.54 mm headers, but also a FTDI USB controller attached to one of the USART ports of the microcontroller. The microcontroller had already a bootloader loaded in it, which made firmware uploading a matter of seconds without the need of a dedicated programmer.

For easier and quicker development of the robot electronics, it was thus decided to use this breadboard and to design a separate PCB with only the necessary circuitry to drive the robot but with header receptacles to plug this breadboard in it.

This breadboard can be seen in the picture below (Figure 3.2):

Figure 3.2 Fraunhofer microcontroller development board

In case of malfunction of the breadboard or the microcontroller, its replacement is as easy as unplugging the faulty board and plugging a new one with the firmware already loaded in it.

The breadboard itself does not only comprise the microcontroller. It already has all the components needed to keep it running and to operate it from a computer. These components are listed here below:

- FTDI USB controller. In short, translates USART signal to USB and viceversa
- 14.7456 MHz crystal, used by the microcontroller
- Access to all ATmega1280 pins via 2.54 mm headers
- 6-pin header for reprogramming
- Mini-USB connector

3.2.1. Peripherals

Despite the numerous peripherals available in the ATmega1280, only a few of them are used. Next list names the actual used peripherals, with a brief description of its implication in the robot operation.

- TIMER1: This timer is used to generate the PWM signal that drives the servo control line (pin PB5). The signal generates a pulse every 20ms, and the length of the pulse goes from 0.6 to 2.4 ms.
- TIMER3: This timer is used to generate an interrupt each 0.25 ms. At this moment, the encoder inputs (Channel A in PD7, Channel B in PE6) are sampled.
- TIMER4: This timer is used as a scheduler. The scheduler has currently two implemented actions. One is to update the positions of the stepper and the servo. The other one is to create and send back the current positions message.
- PCINT0: Pin change interrupt used by the Index channel of the encoder to trigger an interrupt when driven high.
- USART0: The first USART module is used as the communications gateway.
- PORTA: Pins PA4 and PA5 are used to drive two indicator LEDs, red and amber respectively. Red is always on when the program is running. Amber is only on when the microcontroller has received and is processing a new command.
- PORTB has several uses:
    - Pin PB5 is used to output the PWM signal created by TIMER1.
    - Pin PB0 is used by the Index channel of the encoder.
    - PORTD: Pin PD7 is used by Channel A of the encoder.
    - PORTE: Pin PE6 is used by Channel B of the encoder.

- PORTL: Pins PL0 to PL6 are used to control the stepper motor controller.

Table 3.1 shows the overall distribution of the header pins in the breadboard, highlighting the ones that are actually used. The pins are as seen from the top side of the breadboard, as shown in Figure 3.3.

Table 3.1 Microcontroller breadboard pins

	A	B	C	D	E	F
1	VCC	VCC	VCC	GND	GND	GND
2	PG3	PG4	PL0	PB7	PB6	PB5
3	PL1	PL2	PL3	PB4	PB3	PB2
4	PL4	PL5	PL6	PB1	PB0	PH7
5	PL7	PD0	PD1	PH6	PH5	PH4
6	PD2	PD3	PD4	PH3	PH2	PH1
7	PD5	PD6	PD7	PH0	PE7	PE6
8	PG0	PG1	PC0	PE5	PE4	PE3
9	PC1	PC2	PC3	PE2	PE1	PE0
10	PC4	PC5	PC6	PF0	PF1	PF2
11	PC7	PJ0	PJ1	PF3	PF4	PF5
12	PJ2	PJ3	PJ4	PF6	PF7	PK0
13	PJ5	PJ6	PJ7	PK1	PK2	PK3
14	PG2	PA7	PA6	PK4	PK5	PK6
15	PA5	PA4	PA3	PK7	PA0	PA1
16	PG5	RTS	CTS			

Figure 3.3 shows a schematic of the breadboard seen from the top. Pins 5 and 6 in the SV7 connector can be used to reset the device: pin 5 is tied to the $\overline{RST}$, pin of the microcontroller; pin 6 is ground. This SV7 connector is also used for ICSP.

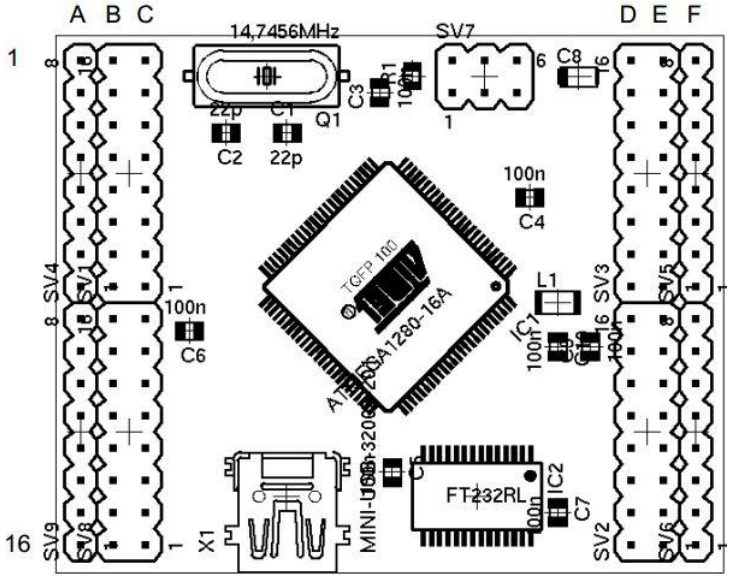

Figure 3.3 Microcontroller breadboard header pins distribution

3.2.2. Bootloader

The bootloader in the ATmega1280 is an AVPROG compatible boot-loader, version is 0.80 beta 3. It uses a modified version of the AVRDude code.

It is configured in a way that it can be accessed during two seconds after power on or reset. To load programs, the AVRBOOT bootloader client is needed in the computer. The bootloader must be thus called by it during this 2 seconds timeout. After programming, the microcontroller automatically resets and after 2 seconds, starts the program.

The microcontroller can be reset by jumping SV7 connector pins 5 and 6 (see Figure 3.4). Pins 1 to 4 are reserved for ICSP programming. The bootloader was programmed into the microcontroller using this header. The arrangement of the pins in this connector is compatible with any ICSP device from AVR. An AVR Dragon STK500 was used for this purpose.

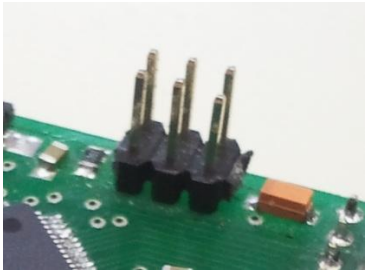

Figure 3.4 SV7 connector detail

3.2.3. Power supply

The low-power ATmega1280 is able to correctly function with a voltage starting from 1.8 V for frequencies up to 4 MHz, 2.7 V for frequencies up to 8 MHz and 4.7 V for higher frequencies. In any case, maximum voltage is 5.5 V.

Consequently, the microcontroller platine VCC pins are powered from a fixed 5 V, 500 mA LDO regulator, which is located in the lower PCB. There is a ferrite in between the regulator and the microcontroller in order to reduce EMI back to the circuit. Refer to 3.2.3 for more information.

3.3. Stepper motor

Pan platform is driven by a stepper motor. Figure 3.5 below shows the Applied Motion's HT17 series stepper motors. The robot uses an HT17-268 stepper, which corresponds to the second motor in the figure below:

Figure 3.5 Applied Motion's HT17 series stepper motors

This section briefly describes how the stepper motor is implemented and operated.

The HT17-268 is a hybrid, bipolar stepper motor. It has the usual 8-wire interface, each of the lines driving half coil. This flexible interface permits using a serial or parallel connection, or even making it unipolar. Figure 3.6 below shows its schematic and its wiring.

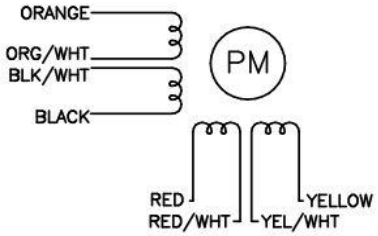

Figure 3.6 HT17-268 stepper wires

As per Figure 3.6, the motor has been wired following a bipolar series configuration. This means that ORG/WHT and BLK/WHT are tied together, as well as RED/WHT and YEL/WHT. Motor terminals are hence ORANGE, BLACK, RED and YELLOW. A series configuration obviously permits the lowest consumption when compared to other configurations, in exchange for a higher overall inductance value.

The HT17-268 is driven by a ST-Electronics L298 which is controlled by a ST-Electronics L297. This combination of L29x circuits makes the motor control very simple. Without the L297, current monitoring and a step state machine would be needed in the processor to control the stepper position. Without the L298, a space-consuming bridge for the motor and its current control circuits would have to be designed.

Figure 3.7 below shows a schematic screenshot from Altium® Designer, showing these two circuits implementation.

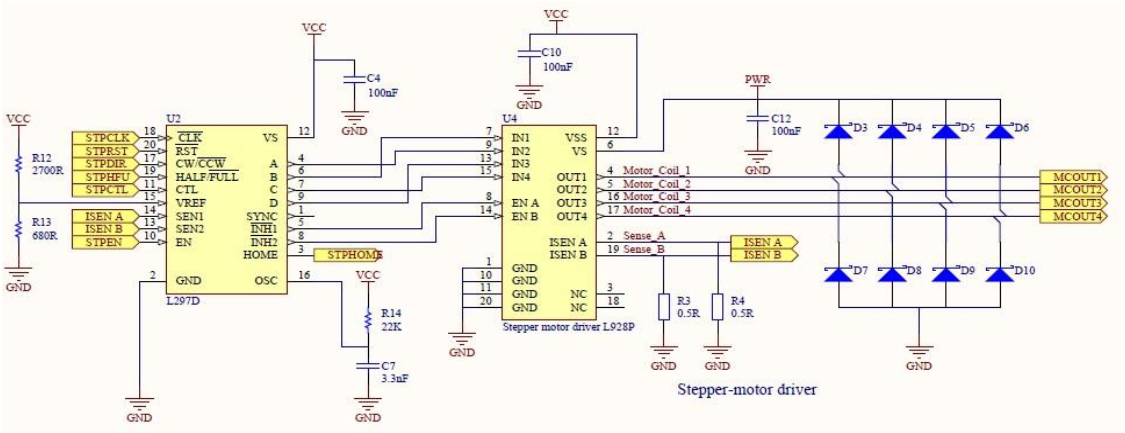

Figure 3.7 L297 and L298 stepper motor control circuits used in the PCB

3.3.1. L297 controller

The ST-Electronics L297 (U2 in **Figure 3.7**) circuit enables the control of stepper motors by only needing clock, direction and mode input signals. With these, it computes and generates the adequate phase states, greatly reducing the amount of code in the microcontroller. The output lines of the L297 are directly connected to the L298, which in essence is a dual full bridge motor driver.

Table 3.2 shows the connections from the microcontroller to the L297 pins, its function and its default value in firmware (at POR).

Table 3.2 Connections between the L297 and the ATmega1280

µC pin	Pin name [nº]	Function	Default
PL0	$\overline{CLK}$ [18]	A low pulse in this pin makes the stepper to advance one step	High
PL1	HALF/$\overline{FULL}$ [19]	Step mode	High
PL2	CONTROL [11]	Whether the chopper should act in INH or ABCD lines	High
PL3	CW/$\overline{CCW}$ [17]	Rotation direction	CW, High
PL4	HOME [3]	Pull-up output. Transistor open when in initial state position	-
PL5	RESET [20]	Resets the state to Home position	High
PL6	ENABLE [10]	Enables the L297	High

To make the controller make the stepper to advance one step (advance one state), the CLK pin has to receive a low pulse of >0.5us. The state change will occur in the rising edge of this pulse.

By default, firmware is programmed to command the L297 to generate phase signals in half-step mode (HALF/$\overline{FULL}$ pin high), thus allowing 0.9º per step precision. The disadvantage of this mode is that in some states of the sequence, one coil becomes totally deactivated, reducing the effective torque/holding force in that position. These moments occur when one of the INHx lines is brought low, as in states 2, 4, 6, and 8 in Figure 3.8. The produced torque in these moments is enough to drive the sensor head without missing steps.

The direction of rotation in this sequence is simply changed by bringing the direction pin (CW/CCW) low.

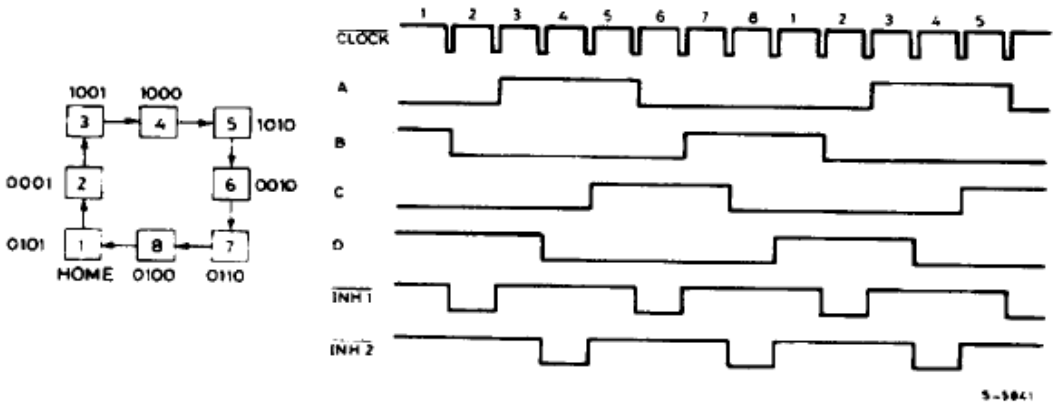

Figure 3.8 L297 half-step states sequence

The VREF pin in the L297 marks a reference for limiting the current in the motor windings. By using sense resistors in series with the motor coils and sensing the voltage drop on these, the L297 monitors this current and acts over the motor lines to control the power dissipated by it and allow over-current conditions. The voltage divider formed by resistors R12 and R13 provides a 1 V reference to the VREF pin. Hence, with the 0.5 Ω sense resistors R3 and R4 translates to a current limit of 2 A, which is almost twice than expected for this motor in this configuration.

3.3.2. L298 driver

The L298 (U4 in Figure 3.7) is a dual full-bridge driver intended to drive inductive loads (such as a stepper motor).

The motor power comes from a separate pin than the logic power supply, allowing higher voltages and currents driving the motor, but also helping reduce the EMC signature to the rest of the circuit. There was a compromise here about how much voltage should be supplied and necessary torque. The motor would provide more torque if a higher voltage level was provided to this pin. However, this would lead to higher consumption and higher heating. For simplifying the overall circuitry, it was decided to use directly the unregulated power coming from the external 5 V transformer, which is also used by the servo power.

Another remarkable topic in the L298 are the so mentioned sense resistors. Recalling what was said in3.3.1, the sense resistors value is 0.5 Ω. Being the voltage reference in the L297 set to 1 V, the current in each winding gets limited to 2 A. Experimental values showed that the motor windings would not consume more than 340 mA each.

3.3.3. Testing

The overall circuitry was thoroughly tested to find faulty conditions. It proved to be highly reliable and robust, and no failures under normal operation were detected thorough the testing.

Tests were basically:

- Trying different CLK pulse widths to control motor velocity. Minimal value was found to be 2 ms between cycles (high-low-high) to avoid the motor missing steps.

- Shorting motor terminals during operation. This just caused the motor to skip steps, or act erratically, without any damage either to it or the driver.

- Theoretically and empirically controlling the temperature of the L298 driver to deduce whether a heat sink was needed. The configuration used on the motor wiring and its consumption at 5V, it was proved unnecessary.

3.4. Servo control

The tilt platform is controlled by a Hitec HS-485HB analog servo (Figure 3.9). This is a low cost, high-torque, karbonite-geared servo. Karbonite gears guarantee that power and precision do not severally suffer from aging and use. Nylon-geared servos are known to have smoother movement, but its drawback is mainly the aging.

Figure 3.9 Hitec HS-485HB analog servo

Servo positioning is traditionally done using Pulse Width Modulation (PWM), and this servo is no exception. This PWM signal is generated from OC1A pin (PB5). The PWM period is 20 ms, which is the standard used by most servos. Figure 3.10 below shows the pulse width values for different arm positions for the HS-485HB.

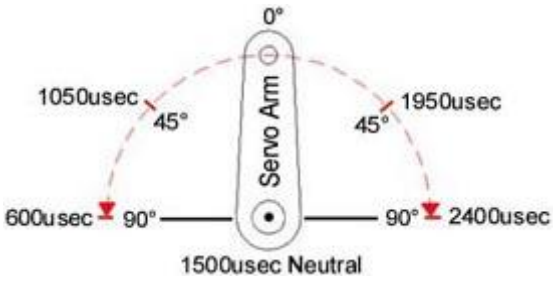

Figure 3.10 HS-485HB range of movement and pulse width

The PWM signal is generated using the TIMER1 Output Compare module, configured in phase-correct PWM mode, a prescaler of 8 and using ICR1 register. Note that ICR1 is a 16-bit register. to define an overflow value. This overflow value is used to generate the period of the waveform. Because the phase-correct mode of the TIMER1 uses a double-slope feature to generate the period (it counts from 0 up until ICR1 value, then it counts down to 0), the value loaded in the ICR1 has to be the desired period divided by two.

In short, the ICR1 value is calculated as follows:

$$ICR1 = \frac{f_c}{prescaler} \cdot \frac{period}{2} = \frac{14.7456 \cdot 10^6}{8} \cdot \frac{20 \cdot 10^{-3}}{2} = 18432$$

The actual pulse width of the signal is defined by the OCR1A register. It works in the exact same manner as the period generation explained before. When the TIMER1 starts counting, the OC1A pin will be high (see TCCR1A.COM1A1:COM1A0 bits in datasheet) until the OCR1A register value is reached. This is called a 'compare match'.

This is better understood by looking at Figure 3.11. This shows the behavior of the OC module when phase correct is selected (TOP value in the figure is the one stored in ICR1 in this case). Note that if the ICR1 value is kept constant, and only the OCR1A register might be modified to adjust the PWM:

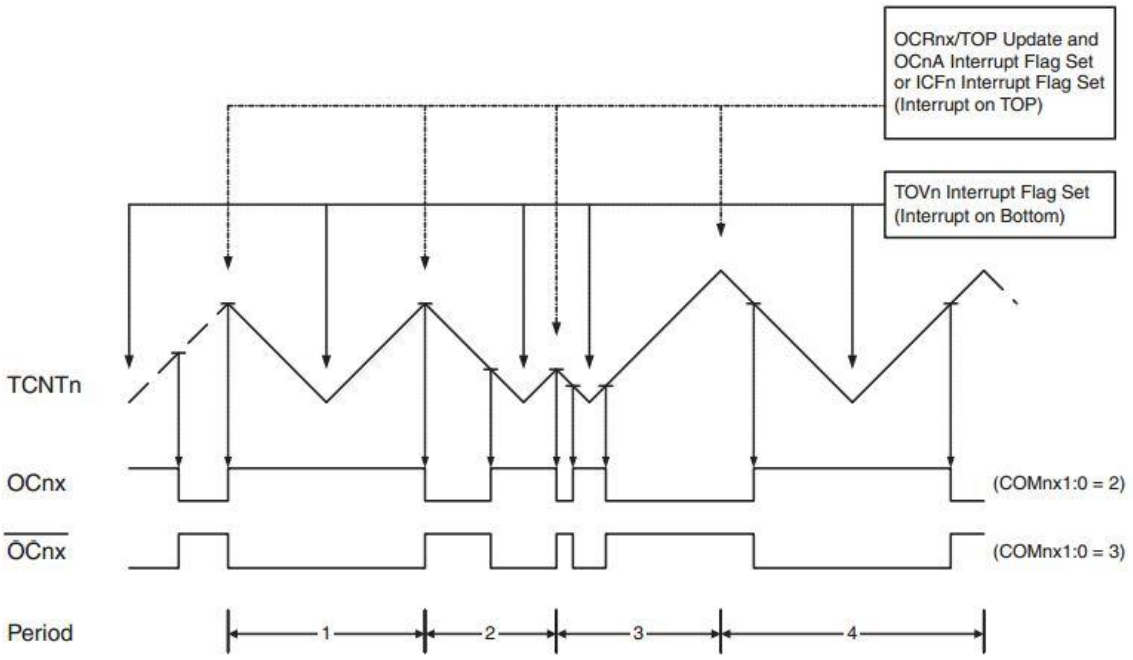

Figure 3.11 Phase correct PWM

Table 3.3 below shows the used registers of the microcontroller and the initial loaded value:

Table 3.3 TIMER1 configuration for correct PWM generation

Register	Brief description	Value
TCCR1A	This register selects the output pin for the PWM signal, and the behavior on each compare match	0x82
TCCR1B	This register is used to select the prescaler	0x12
ICR1	Stores the desired period value	18432

Refer to 4.3.1 to know how the OCR1A value is modified and how the final desired signal is generated in code.

3.5. Power supply

Power supply to the robot comes from an external transformer. This transformer is connected to the electronics box by means of a DC-connector, where it gets regulated. This entire process is detailed in Figure 3.12 below:

- Pseudo-regulated power coming from the external 220 V transformer
- Regulated power from a LDO power regulator
- Ferrite-isolated power for the microcontroller

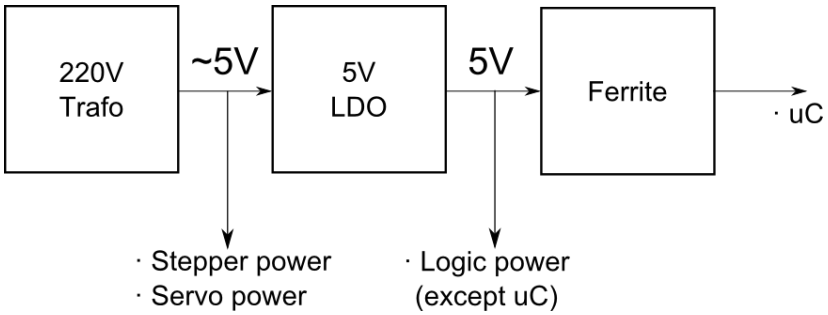

Figure 3.12 Power distribution in PCB

3.5.1. Input stage

The robot is intended to be powered from 220-240 V wall plugs, using a transformer to transform to 5 V DC. The transformer nominal current rating must be of at least 2 A (1.4 A are rarely overcome).

Most 5 V transformers use a DC-power type connector with a central pin diameter of 2 mm. The PCB uses a Kykon KLDHCX-0202-A-LT connector, which is compatible with any connector of this type, including locking-type.

Non-logic electronics are fed straight with the transformer power: The stepper motor receives power from the L297 “non-logic” pin, which is in tied to the transformer power rail. The servo power also comes from the transformer, without further regulation. Protection TVS diode is provided just after the connector, which breaks if a voltage above 16V is applied at the input.

3.5.2. LDO power regulator

For the logic circuitry, power is regulated to a fixed 5 V by an ON Semiconductor NCP4629 in a DPAK-5 encapsulation. This is capable of driving up to 500 mA. The PCB provides a small plane to act as heat dissipator.

Despite this, measurements indicate an average circuitry consumption of less than 200 mA, and thus no heat-sink is needed (in this conditions, junction temperature at ambiance is calculated[1])

Figure 3.13 below shows the power stage in the robot. Note the LDO at the right half of the scheme, with its decoupling capacitors:

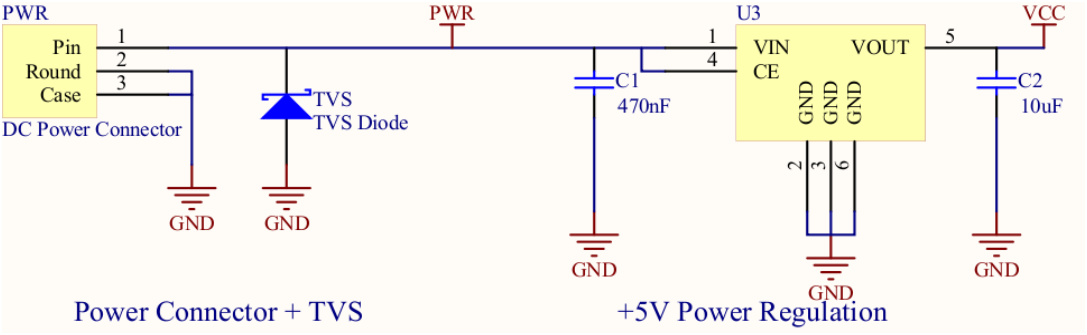

Figure 3.13 Power stages

3.5.3. µC Power isolation

Digital noise from µC is avoided to filtrate into the rest of the PCB nets by isolating the µC power rail with a ferrite. This is a common procedure in any analog/digital circuit (meaning that there exists analog to digital conversion), although it could have been skipped in this almost purely digital PCB.

The ferrite inductance value is low enough to avoid important voltage drop in DC, being this less than 0.1 V (measured). The ATmega1280 can work with voltages as low as 4.7 V when operating at 14 MHz.

3.6. Encoder signals

An encoder is used to retrieve a more precise position of the stepper motor. This can be useful, for example, to detect incongruence between theoretical stepper step count and encoder step count, which could occur if the stepper is moved externally. In this case, the stepper step counts would remain the same, but the encoder would then report that the encoder is actually another position, which can be detectable by software.

[1]From NCP4629 datasheet, it is calculated to be 53ºC. Limit is 150ºC.

The used encoder in the robot is an Avago HEDS-5540, which has 3 channels: two for position (CHA and CHB) and one for index signal (CHI). It can be seen in the Figure 3.14 below:

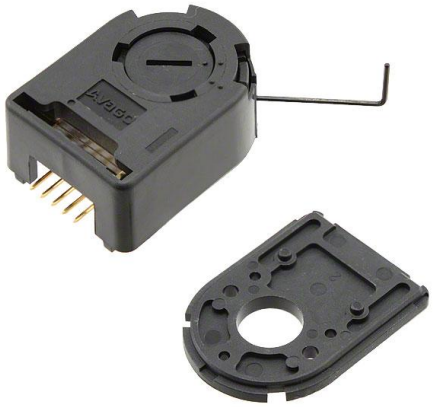

Figure 3.14 Avago HEDS-5540 3-channel encoder and its cover

Moreover, more important is that the encoder has an Index signal (CHI) which is present once at each rotation. This signal is a high pulse of duration 1/4 of cycle (Figure 3.15), which is interfaced directly to an external interrupt pin of the microcontroller (PCINT0 pin). Each time the encoder is rotated through this Index, it will send a pulse to the microcontroller, causing an interrupt and allowing to effectively detecting that a rotation has been made. In fact, this is the signal used to calibrate the pan platform.

The encoder's position signals (CHA and CHB) are quadrature signals, which have to be decoded. The decoding is simple: quadrature signals can be simply seen as a binary Gray-coded signal. This decoding is again done in software, by quickly sampling both signals, comparing its value with the sample before, converting it to binary and, if applies, increasing or decreasing a dedicated counter. The advantage of using quadrature signals is that direction can be easily extracted from them.

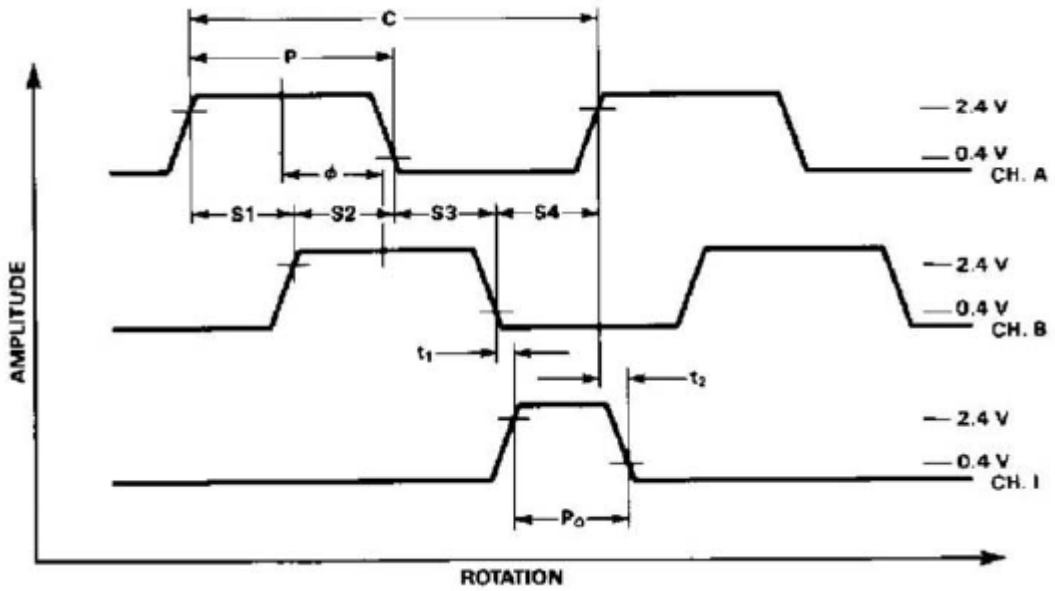

Figure 3.15 Encoder output waveform

If CHA is leading CHB, like in the picture, means that the direction of rotation is counter-clockwise, looking from the encoder end of the motor.

The electronic interface of the channels with the microcontroller only requires the use of 2.7 kΩ pull-up resistors, as seen in Figure 3.16 below:

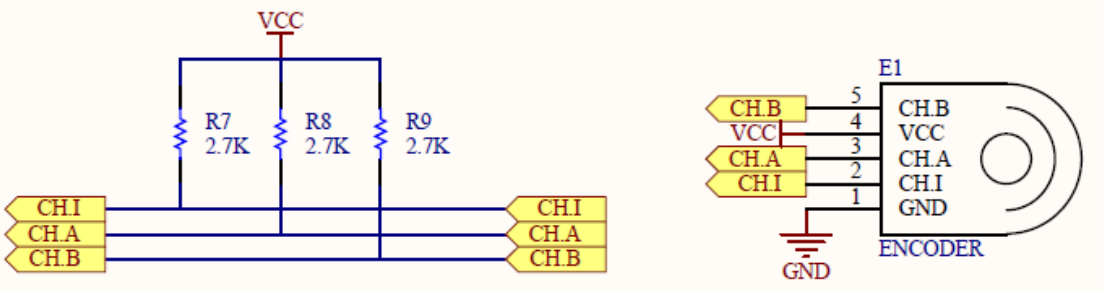

Figure 3.16 Encoder interface

3.7. Connectors and wiring

Wiring in the robot is simplified to a single cable running from the electronics box up to the movable platform. Fail-safe connectors are provided in the wiring, so connection is always made in the correct way.

Figure 3.17 shows the blue header pins of the PCB. The shorter, thicker line at the left is the small tip in the blue header.

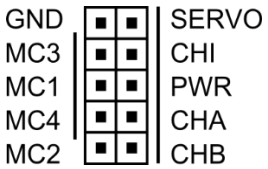

Figure 3.17 Header connector pins in PCB

An intermediate flat cable is used to run from this header to the exterior of the electronics box. From here, it is connected to a shielded cable that runs up to the head and connects to it. Wiring changes between cables are shown in the correspondent Annexes section.

The connectors between each intermediate cable are Molex Micro-Fit 3.0 connectors. These are 3.00 mm pitch, high-density, low-power connectors (stand up to 5 A per wire). They are perfect both for power and signal management.

3.8. Power connector

The power connector, where the transformer output is plugged, is a Kykon KLDHCX-0202-A-LT. It has the usual dimensions for DC-power connectors used in most 5 V transformers. Figure 3.18 below shows its connections.

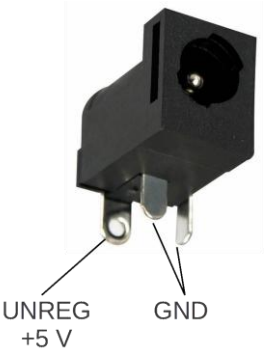

Figure 3.18 Power connector wiring

4. ROBOT FIRMWARE

This chapter describes how the robot firmware is structured, and how to operate it without the dedicated ROS driver, i.e. straight from a terminal program.

4.1. Communications protocol

The Robot is connected to the host computer with a USB cable. It uses a dedicated FTDI IC to decode the USB data and translate it to serial USART data, which finally is processed by the microcontroller. Communications with the robot are based in a simple messaging protocol. In short, the user has to send a key ASCII character followed by data to the robot, and this will answer with a specific response depending on the request.

The robot uses the following serial port configuration:

- Baud rate: 9600 bps
- Data bits: 8
- Parity: Even
- Stop Bits: 1
- Handshake: Not implemented
- End of line character: <LF>

4.2. Firmware description

The firmware code behavior is quasi-deterministic, as most of its timing is based in a scheduler (using timers).

Besides the main routine, which is an endless loop which waits for data to be available in the USART (character reception is also interrupt-based), TIMER4 is used to drive an interrupt-driven scheduler. This scheduler has two functions. First, it computes if it is necessary to move the stepper (pan) and/or the servo (tilt). Secondly, if allowed, it sends their positions to the USART. These rates are adjustable by code.

4.2.1. Main code

The main program proceeds as follows: As soon as the robot is powered on, the microcontroller goes through peripheral init routines, which include setting up

the USART port, the servo PWM controller, the stepper motor controller configuration, the LEDs, the scheduler and the interrupts. After running these init routines, the program enters an endless loop. The program is brought out from this state as soon as it receives a command from a host computer via the USART port, which causes an interruption on the USART port and the loop reads the content in the receive buffer. Figure 4.1 shows this process.

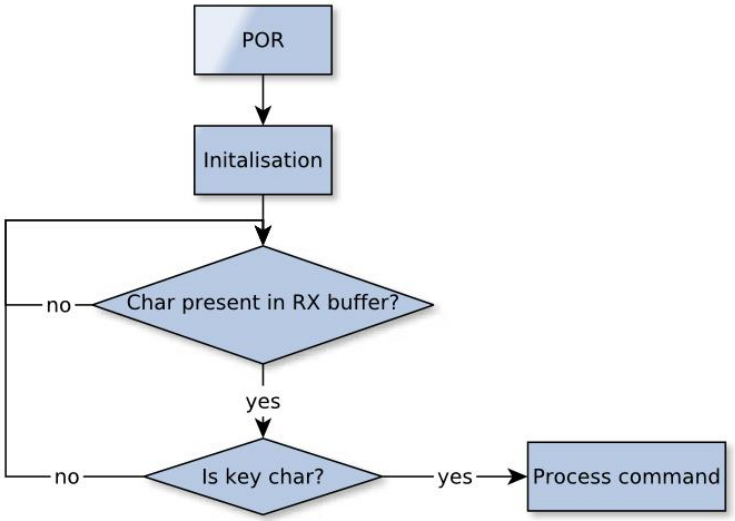

Figure 4.1 Power on and main routine

4.2.2. USART module

An interruption is called each time a character is received through the USART port. The USART port routines implement a configurable-size circle buffer for both reading and writing buffers. When the interrupt is called, the incoming character is copied to the circle buffer; then it returns from interruption.

At this point, the condition in the endless loop will detect that the circle buffer is not empty and will extract the corresponding character from the circle buffer, to finally detect if it is a command character (see Figure 4.1). In that case, the specific routine will be called, reading the subsequent received data characters and correctly executing the order after having received an entire message (which ends with <LF>) after returning to the main loop.

4.2.3. Scheduler

Recall the scheduler had two functions:

1. Repositioning the stepper and/or the servo if necessary.
2. Gathering current positions and reporting them.

The scheduler uses TIMER4 configured in CTC mode to create interrupts each 1 ms.

4.2.3.1. Repositioning the platforms

In order to achieve a smooth and synchronous movement between the stepper and the servo a special procedure had to be done here.

The servo does not give position report, having to rely that the position commanded to the PWM controller will be the one immediately taken by the servo. A workaround on this was made in code, in order to discretize each step of the servo to make it known and fixed: A 256 positions lookup table is available in code, and each time a position is commanded to the servo, the code will command a progressive advance, step-by-step through this table, instead of commanding a direct PWM value to the target. Commanding the servo in this "step-by-step" mode obviously permits knowing on which step the servo is currently located, but it also permits adjusting the delay between each step -and thus being able to adjust its rotation velocity.

Similar methodology is used in the stepper, with the advantage that the stepper only requires a pulse to be sent to the L297 controller. The pace of these pulses will determine the velocity of rotation of the stepper. Because the minimum space between pulses for this stepper not to miss steps is 2 ms, and that the "reposition" interrupts are created every 1 ms, the maximum rotation speed of the stepper can be obtained by:

$$v_0 = \frac{0.9º}{1ms + 2ms} = 300\frac{º}{s} = 5.23rad/s$$

The stepper and the servo have two "pendant steps" counters in software. Every time a position is received through the USART, the code will compute the difference between both current positions and the target positions, effectively knowing how many steps are pendant. The counters are then updated. From here, at each scheduler interrupt (which occurs every 1 ms), if there are pendant steps for either the stepper or the servo, they are commanded to advance one step. To adjust when to do a step, a second "latency" counter is used. When in the interrupt, only if the latency equals the desired value a position advance will be commanded. Of course, in this case the disadvantage is that the best resolution that can be achieved for regulating this delay is 1 ms. Nevertheless, in order to avoid skipping in the stepper, its inter-step time must be above 3 ms when loaded, so this resolution is enough for this application.

Proceeding with interrupts to move permits moving them both virtually at the same time, for several positions, without even consuming process time or having to wait for one to reach is position before moving the other. The alternative (and first approximation) was simply to command a step jump, then

delay; command another step jump, delay again... which resulted on the program having to wait for all the steps to be done to continue.

The code below shows an excerpt of how the step advance is done:

```
if (rem_steps > 0)
{
     if(count == lat )   //  Yes,  it's  time  to  move  once

     {
          // Command a single advance
          advance();

          // Update pending steps
          set_rem_steps( rem_steps - 1 );

          // Reset counter
          count = 0;
     }
     else
          count++;  // Not yet but just keep counting
}
```

4.2.3.2. Gathering positions

The second function of the scheduler is to report both stepper and servo instantaneous positions. These positions are gathered and reported each 25 ms (40 Hz), so each twenty-five TIMER4 interrupts. A counter in code is used to count the interrupts, so the pace of this gathering is adjustable in the #define SCHED1 in *scheduler.c*.

A flag (*sl1_isEnabled*) enables or disables this gathering of positions to be reported to the USART. This flag is activated by the host computer by sending a <R><1><LF> string. To deactivate the flag, just an <R> has to be received.

4.2.4. Stepper control

The stepper is controlled by the L297, which requires some inputs from the microcontroller. These inputs are simply pins tied to standard PORTL pins of the microcontroller. A list of these pins can be seen in Table 3.1.

4.2.4.1. Step generation

A single step is generated by driving the PL0 pin low for more than 0.5 µs. However, to avoid skipping, this value is set by software to be 0.9 µs. A single step generation is made by calling:

```
stepper_advance();
```

This method is defined in *stepper.h* header file. A part from advancing, this method also is in charge of increasing or decreasing the step counter, depending on the rotation direction.

The direction of rotation can be adjusted with the PL3 pin. A high signal in this pin will set the direction of rotation to clock-wise. A low value will set it counter-clockwise. There is a dedicated method for this, and for directly obtaining the current rotation direction by hardware (the pin status).

```
stepper_set_direction();
stepper_get_direction();
```

4.2.4.2. Stepper calibration

It is important to note that before any pan position command is sent, it is mandatory to first calibrate the pan platform to safely operate the robot. By doing this, the pan platform rotates counter-clock wise until it detects an Index flank from the encoder, which is sent once per each encoder complete rotation. This enables the firmware to have a known reference location for the pan, allowing knowing the real position and avoiding turning it further than the tilt servo and sensor wires permit. The Index position can be seen in Figure 4.2. Once the calibration reaches this position, the pan moves 4 steps back to enable easy recalibration (if not, the platform would make an entire rotation again, probably twisting the wires too much).

After successful calibration, both stepper and encoder report a position value of 0. The robot can be now normally operated.

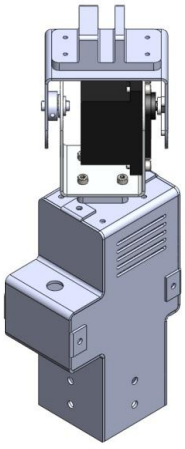

Figure 4.2 Robot pan situated at “Home” position

4.2.5. Encoder signals analysis

Recall encoder signals are a train of pulses in two quadrature channels. These permits the extraction of the rotation direction, for example, by counting pulses

(interrupt when a change in level is detected) and caring about which one of the channels is “in front” of the other. This is the most obvious approximation, but can lead to glitches in pulse count.

A glitchless approximation is to consider these quadrature signals as a Gray code signal, which, when read, is compared to the immediately last value to extract direction of rotation and thus a single counter can be used to keep track of the position. This can be seen in Figure 4.3, where encoder signals are seen as Gray code.

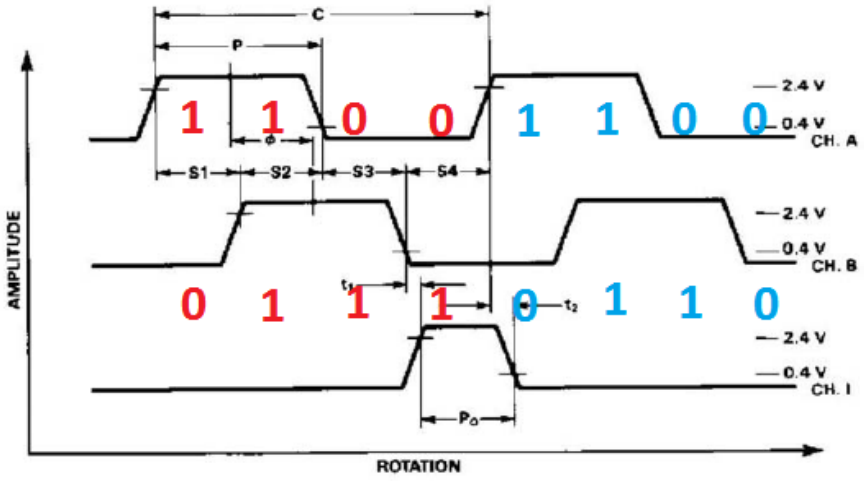

Figure 4.3 Encoder signals seen as Gray code

A dedicated timer (TIMER3) is used to query a sampling of the encoder channels at an adequate sampling rate.

Consider that maximum stepper step is done every 3 ms; 1 stepper step are 5 encoder steps, so an encoder step will occur every 0.6 ms; considering Nyquist (with a factor of 2.5):

$$Sampling\ Rate = \frac{0{,}6\ ms}{2.5} = 0.24\ ms/sample$$

4.3. Protocol Description

Communications with the robot are based in a simple querying protocol. In short, the user has to send a key character followed by data to the robot, and this will answer with a specific response.

The header characters for each function of the robot are shown in Table 4.1 below.

Table 4.1 Possible queries from host to robot

Description	Header char
Set pan to position	P
Stepper advance one step	U
Set tilt to position	T
Retrieve pan and tilt positions	R
Pan calibration	L
Stop movement	E
Flush USART buffers	B
Loopback mode	C
Retrieve robot name	N

4.3.1. Pan positioning

Pan platform is driven by a 200-step stepper motor, that is, 1.8º per step. However, the motor is driven by firmware in Half-Step mode, thus allowing 400 steps and a precision of 0.9º per step. Moreover, an encoder is attached to the stepper motor, allowing further verification of its position. The encoder acts only as monitoring/verification device, as the position is computed by counting the stepper motor steps. It can thus be used to verify if the steps count and its reported position correspond. This is useful to detect if the pan platform has been moved from its calibrated state, for example, because of unexpected external movement of the pan platform.

Summarizing, the message is composed by:

1. Key character 'P'
2. ASCII characters representing a signed integer with the target position
3. Comma character ','
4. ASCII characters representing an unsigned char with the latency in ms between steps
5. End of line

Figure 4.4 shows the basic software positions of the pan platform. Note that the position is sign sensitive, meaning that a -400 will rotate the stepper 360º in CW direction from the index position 0.

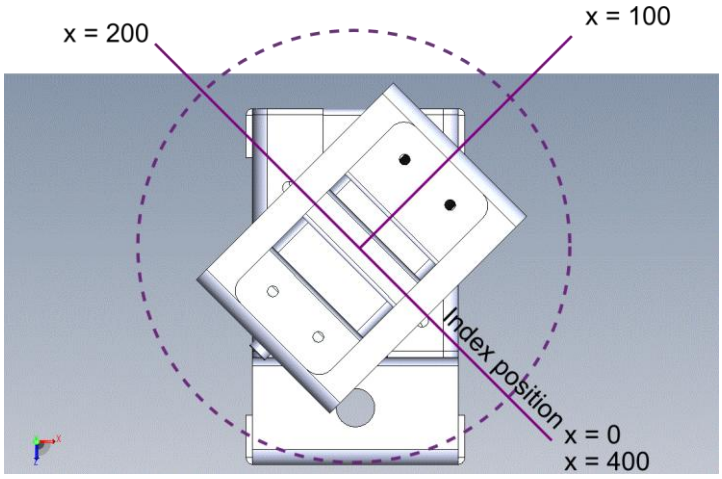

Figure 4.4 Stepper positions in software

E.g.: The user wants to fully rotate the pan (360º = 400 steps) from “Home” state, with latency between steps of 10 ms:

```
<P><4><0><0><,><1><0><LF>
```

Once the message has been sent, the robot starts acting the pan platform. The stepper jumps between steps every 10ms. When the final position is reached, a message with final stepper and encoder positions is sent. This message is formatted as shown below:

```
<P><p_MSB><p_LSB><e_MSB><e_LSB><LF>
```

being p_MSB and p_LSB the most and less significant bytes of the stepper position and e_MSB and e_LSB the most and less significant bytes for the encoder position.

Finally, a message containing an <ACK> character will be sent to indicate that the request is complete.

4.3.2. Pan single step advance command

By using this command, the robot pan platform can be advanced a single step in either direction. The direction of rotation is sent in the body of the message, as specified in the table below (Table 4.2):

Table 4.2 Direction of rotation byte

Byte value	Direction of rotation
0x00	CCW
0x01	CW

The message is composed by:

1. Key character <U>
2. Direction of rotation (Byte)
3. End of line

4.3.3. Tilt positioning

Tilt platform is driven by a Hitec HS-485HB servo. Its 180º movement is discretized in 256 steps, hence having a precision of around 0.7º per step. The servo has an internal closed-loop. The advantage of this is that no external loop control circuitry is needed -as was in the stepper with the encoder-, simplifying the control burden. The disadvantage is that the servo does not report its actual position, having to rely that the servo closed-loop is operating correctly. However, this can be assured reliable enough to ensure precise positioning.

Figure 4.5 below shows the basic servo positions in software:

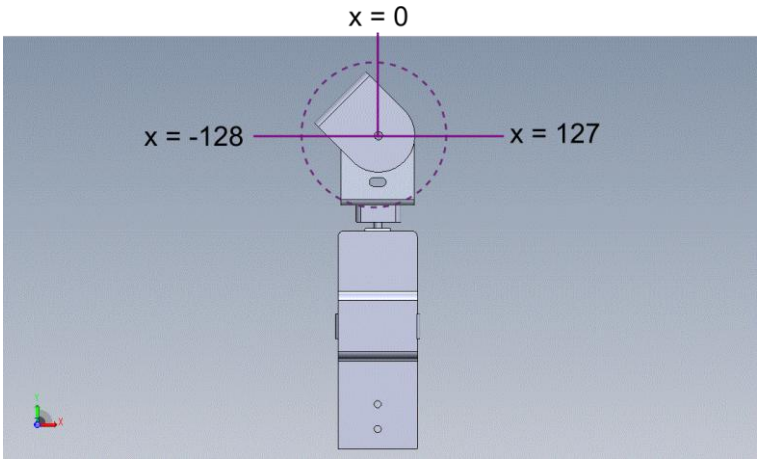

Figure 4.5 Servo positions in software (robot seen from left side)

Despite these physical differences with the stepper motor, their behavior and control was made to be similar. Hence, the message is again composed by:

1. Key character 'T'
2. ASCII character with the desired position.
3. Comma character
4. ASCII character with the desired latency.
5. End of line

This is an example message:

```
<T><1><2><8><,><1><0><LF>
```

Once the message has been sent, the robot starts acting the tilt platform. The robot will report each intermediate position. When the final position is reached, an <ACK> character will be sent.

4.3.4. Position retrieving

There are two ways of getting the immediate positions of the robot.

- Continuous: report is done every 25 ms (40 Hz)
    - To activate:

```
<R><1><LF>
```

    - To deactivate:

```
<R><LF>
```

- Single: report is done once after key character is received

```
<r><LF>
```

The robot will then respond with the pan position (ASCII), followed by the encoder reported position (ASCII) and finally tilt position (ASCII), each separated by commas. After that, a <ACK> message is also sent to indicate request completion.

4.3.5. Pan platform calibration

As stated in 4.2.1, when the system is started up, the robot does not know the actual position of the pan platform. Hence, a calibration process must be run at

least once after power-on. This must be requested by the host computer at least once.

The pan platform calibration is requested by simply sending:

```
<L><LF>
```

The robot will self-calibrate, and finally respond:

```
<L><ACK><LF>
```

4.3.6. Immediate stop

While robot is performing a movement, it can be ordered to immediately stop and wait for other instructions. This is done at any time by sending an ‘E’ ASCII character, followed by a <LF>.

```
<E><LF>
```

4.3.7. Other messages

4.3.7.1. Retrieve robot name

Simply return the name of the robot in ASCII characters. Answer should be «COB3DMD» plus <LF>.

4.3.7.2. Serial port loopback mode

In this mode, the robot will just echo back any received characters, except for 'q', which is used as escape character.

To enter this mode, send:

```
<C><LF>
```

The robot will respond <C><ACK><LF> after 'q' escape character is received.

4.3.7.3. USART transmit and receive buffers flush

Transmit and receive circular buffers of the USART can be flushed at any time by sending a ‘B’ character.

5. ROS PACKAGE

Since the target user of the robot is the people in charge of Care-o-Bot development, integration with the already existent ROS COB stacks was a requirement (instead of developing an entire standalone controller for it). For that reason, the only difference between using the actual COB or the Mapping Demonstrator packages is that, in the moment of “bringing up” the robot, instead of calling e.g. the Schunk Powercube® drivers, it will call this dedicated *cob_3d_mapping_demonstrator* driver.

5.1. Introduction to ROS

ROS is an open-source, meta-operating system or middleware for robotics. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

The ROS runtime "graph" is a peer-to-peer network of processes that are loosely coupled using the ROS communication infrastructure. ROS implements several different styles of communication, including synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server. ROS currently only runs on Unix-based platforms.

5.1.1. Features

- Thin: ROS is designed to be as thin as possible. Code written for ROS can be used with other robot software frameworks. A corollary to this is that ROS is easy to integrate with other robot software frameworks.
- ROS-agnostic libraries: the preferred development model is to write ROS-agnostic libraries with clean functional interfaces.
- Language independence: the ROS framework is easy to implement in any modern programming language. Implementations in Python, C++, and Lisp, and experimental libraries in Java and Lua already exist.
- Easy testing: ROS has a built-in unit/integration test framework called *rostest* that makes it easy to bring up and tear down test fixtures.

- Scaling: ROS is appropriate for large runtime systems and for large development processes.

5.2. Communicating between the ROS network

There are several ways of communicating ROS elements. Most of them are message based, but depending on which type of message used, the communication methodology can be greatly different.

In short, one can distinguish between:

- Publisher/subscriber method

This method uses a *msg* file. *msg* files are simple text files that describe the fields of a ROS message. They are used to generate source code for messages in different languages.

This method is really useful to communicate non priority data, high throughput data, such as the positions of the robot. The program will continue working without any notice either if one of these messages is lost.

- Server/client method.

This other method uses a *srv file. srv* files describe a service, which is essentially the same as a *msg* but with two fields, a request and a response.

The server/client method is more useful for event management, for example. It is mainly used thorough the Care-O-Bot packages as a way to enable or disable the main functionality of a package.

5.3. Package outline

Because of the confidentiality policy at the Fraunhofer IPA, not much detail can be given on this. To put in context, though, next figure below (Figure 5.1) is used to show an approximation to the ROS Package structure and the information/command messages:

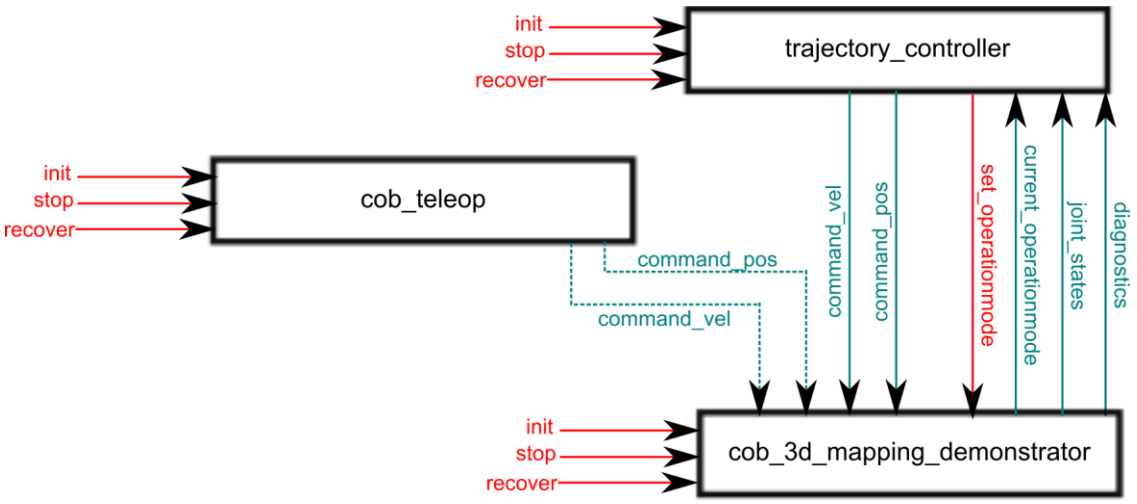

Figure 5.1 Care-o-Bot® 3 simplified ROS Package structure (Published messages are in blue. Service messages are in red.)

5.3.1. *trajectory_controller* package

The *trajectory_controller* package computes, from a final desired position or positions, all the intermediate movements for the robot to follow. It is proprietary of the Fraunhofer IPA, and it is still under development.

The output messages (publishes) are position (with intermediate positions at each publication) or velocity (immediate velocity that the robot must take until next publication). It uses a physical description file (URDF file) which is specific for each robot (see 5.5).

Short after starting, it signals whether it will publish position or velocity commands. At the same time, it will have to receive confirmation from the *cob_3d_mapping_*demonstrator package. Also thorough the operation, it has to receive feedback/report information from it.

5.3.2. *cob_teleop* package

The *cob_teleop* acts as a direct input for manually moving the robot using an external joystick. It directly publishes the *command_pos* or *command_vel* messages bypassing the trajectory controller, so to be read by the *cob_3d_mapping_demonstrator.*

5.3.3. *cob_3d_mapping_demonstrator* package

This is the main driver created for the robot.

The *cob_3d_mapping_demonstrator* receives the position (or velocity) commands (see 5.4) and queries and controls the robot as specified in 4.3. It uses the parameter server to configure the startup and operating conditions.

As output messages, it continuously publishes the current operation mode, set by the *trajectory_controller* in the beginning. This can be changed at any time. It also publishes a joint_states message, which is composed by:

```
string[] name       // The name of the joint being reported
float64[] position  // Its position in radians
float64[] velocity  // Its last velocity in rad/s
float64[] effort    // 0, not used in the robot
```

It also publishes a diagnostics message, which contains status and health data, such as if the stepper and the encoder are reporting congruent positions (see 4.2.5).

5.4. Modes of movement

As stated, there are two modes of movement: by position or by velocity.

The position mode receives a target position from the upper levels, such as the *trajectory_controller* package or simply an input in console. Similarly, the velocity mode receives a rotation velocity command at which the platform is required to turn. This last can be easily understood by imagining the user using a joystick or a gamepad to move the platform. Here, moving the pad does not create position commands, but velocity commands: the more the pad is pushed, the more fast the platform has to rotate.

Whatever the mode is, the input position or velocity is evaluated before being executed. The code takes the maximum and minimum limits data from the URDF file.

5.4.1. Position command

Really simple to understand is the position mode, where the user simply has to specify a target position. The algorithm will start moving the platform at a constant speed until the target position is reached. This rotation speed is specified in the parameter server file as *const_speed*.

5.4.2. Velocity command

In the velocity mode, more complicated processing is done. It has to be noted that the code had some glitches which could not be resolved in time. Nevertheless, it operated correctly most of the time.

Mainly, what the velocity mode does is to take the difference between the current time moment and the moment where the last velocity query was received. With this time difference, and the desired rotation speed, the next target step can be integrated. The rate of the publisher is specified in the parameter server as *frequency.* It is easy to see that glitches in here can appear easily. For example, if the time gap is too large, the computed velocity would be also too much. Other glitchy moments are identifiable by really small instants when the platform smooth movement gets interrupted. Small delays in the ROS code as well as in the microcontroller can lead to small hangs of the program, which sharpen the movement. Further work to enhance stability here is required.

5.5. URDF files and transformation

URDF files are a new formatted description file that can store a physical description of a robot in a standardized way, in order to enhance interchangeability and code reuse.

URDF are based in the so-called links and joints. A link can be seen as the rigid part of a robot arm, for example. A joint would thus be the mobile joint making union between two links (see **Figure 5.2**).

These files are XML based, and some of them can use an enhanced version of the URDF format (which is easier to understand and write) called XACRO.

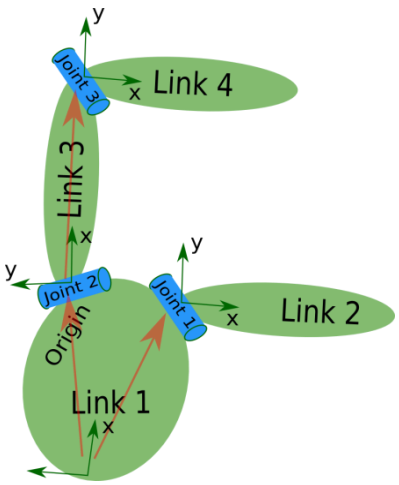

Figure 5.2 URDF links and joints proposal

The ROS *rviz* package can be then used in conjunction with this to obtain 3D, axial representation of the robot. This is interfaced with the position reported by the robot, thus getting a realistic robot transformation which is directly interfaced with the point-cloud from the sensors.

5.6. Parameter server

Parameter server is loaded when ROS core is started. The elements that are usually loaded to it are some configurable parameters of the robot, such as the serial device where it is connected and its speed, the joint names (these cannot be changed), velocity limits for each joint, position publication rate, etc. These parameters are read from an XML or a plain text file, which has to be referenced in the package startup configuration file, common in all ROS packages.

Next frame below shows an example parameter server file to load.

```
serial_device: ttyUSB0
serial_baudrate: 57600
joint_names: ['stepper', 'servo']
fixed_velocities: [1.02, 1.02]
auto_initialize: true
operation_mode: velocity
min_publish_duration: 0.01
frequency: 50
```

6. CONCLUSIONS

The work started on September 15th, being the robot delivered on March 25th. Everything was done from scratch, from conception to realization, as seen through the document: creating a structure, creating an effective moving platform, selecting materials and components, creating the electronics, soldering, assembling, testing and finally optimizing. It has been a hard job, but the experience I got from this hard work has effectively been more than I could expect.

As a conclusion to this document, here are some points learnt during the development, and also future enhancement proposals are stated:

- Concerning structure:
    - The supporting structure was flawless, but some points shall be refined such as scratches that appeared during testing due to misuse and sharp corners that may lead to cuts during maintenance disassembly. The Alucobond® plates used to cover the structure can get easily scratched, and thus it is recommended to use another harder material.
    - The designed head is easily disassembled and reassembled, with no need of any recommendations. The process can be done in any order.
- Concerning electronics:
    - Electronics proved totally trouble-free and reliable. These where extensively tested against failure.
    - The stepper motor used moves the sensors correctly and without missing steps. However, a slightly more powerful stepper engine might be better for heavier loads. Depending on the motor, this would require electronics rework.
    - At the end of the project, the electronics box was relocated from the moving head to the base, as was requested by the robotics team leader. This was a proposal to enhance visual appearance, and was requested at the very last moment.
- Concerning software:
    - The firmware code is rich in feature, reliable and performing. It was done with optimization in mind, but without making it hard to understand. Difficult points were correctly documented in code.

- The ROS package was a hard point in the project. Due the simplified functionality of the demonstrator compared to the other robots at the Fraunhofer IPA, the code had to be adapted to the existent complex software, this proving difficult in some situations, as the code had to emulate some unavailable functionalities in order to work.

- The position command works without flaws, as well as the position error reporting. ROS easily turns on an error flag when this occurs, and the entire system and user get warned.

- Despite the efforts, the velocity command was still glitchy at the moment of delivery. More work is required in this point.

A functional and robust robot has been built, as stated in the main requirement. The robot has been thoroughly tested by the project manager and it was approved to use it as a technology demonstrator at the *AUTOMATICA* fair in Munich.

7. BIBLIOGRAPHY

[1] Application note: Atmel AVR4027, “Tips and Tricks to Optimize Your C Code for 8-bit AVR Microcontrollers”, 2011

[2] Application note: Atmel AVR1200, “Using external interrupts for megaAVR devices”, 2011

[3] Application note: Atmel AVR360,“Step Motor Controller using tinyAVR and megaAVR devices”, 2003

[4] Application note: Atmel AVR040, “EMC Design Considerations”, 2006

[5] Application note: Atmel AVR186, “Best practices for the PCB layout of Oscillators”, 2010

[6] Application note: Atmel AVR130, “Using the timers on tinyAVR and megaAVR devices”, 2002

[7] Application note: Atmel AVR131, “Using the 8-bit AVR High-speed PWM”, 2003

[8] Application note: Atmel AVR244, “UART as ANSI Terminal Interface”, 2003

[9] Application note: Atmel AVR4029 “AVR Software Framework - Getting Started”, 2012

[10] Application note: ST Electronics ST AN470, “THE L297 STEPPER MOTOR CONTROLLER”, 2003

[11] Encoder sig. treatment: http://www.mikrocontroller.net/articles/Drehgeber

[12] AVR Dude bootloader:http://www.mikrocontroller.net/articles/AVRDUDE

[13] Stepper motor basics: http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf

[14] L297 and L298 basics: http://www.roboternetz.de/community/threads/773-Schrittmotor-ansteuern-mit-L298-und-L297