Robocup Junior 2016

Soccer Light Robots

Team “SPQR”

ITIS G. Galilei Rome

Technical Documentation

Team members

Team Members

Enrico Di Claudio

Age: 17

Hobbies: Electronics, Bike, Sports in general

Giulia De Iulis

Age: 18

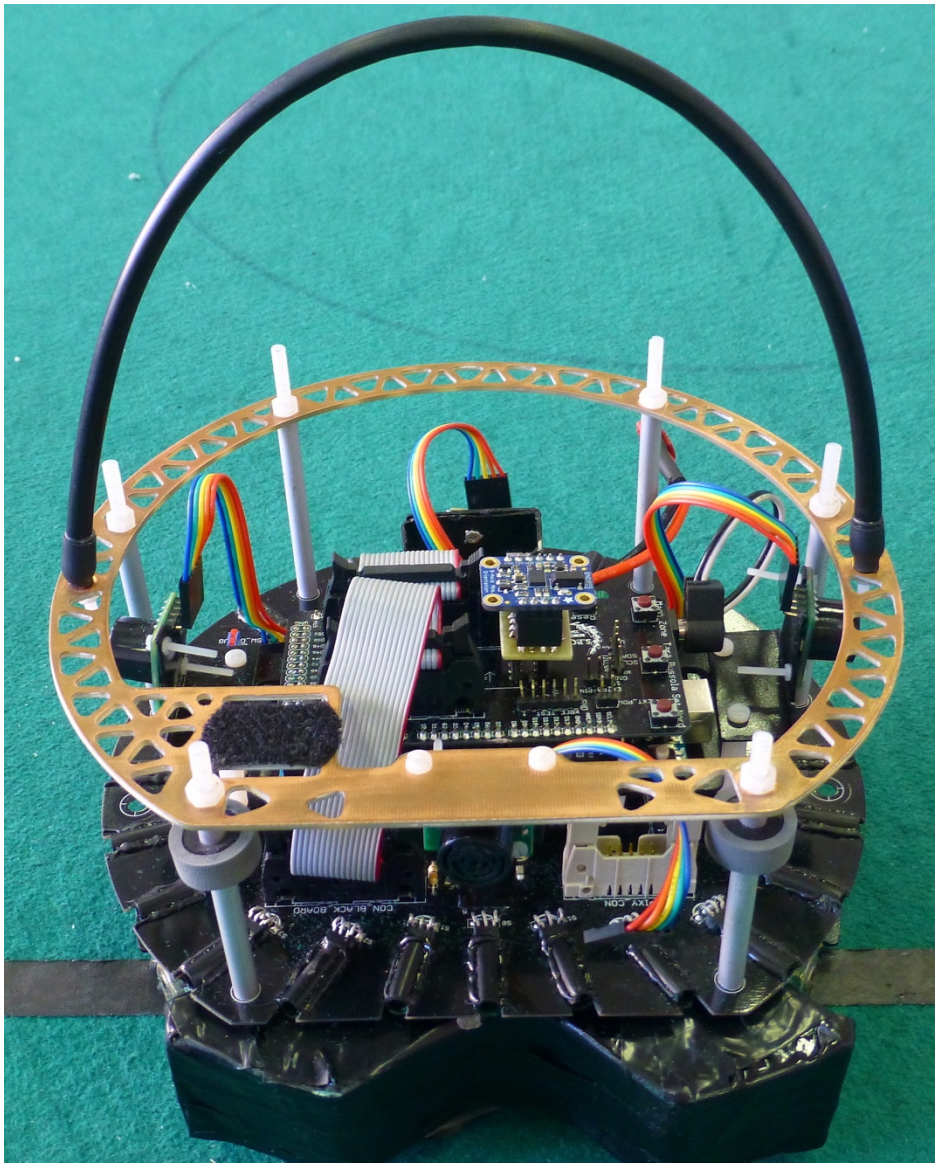
Hobbies: Astronomy, Reading, Playing piano

Flavio Galasso

Age: 17

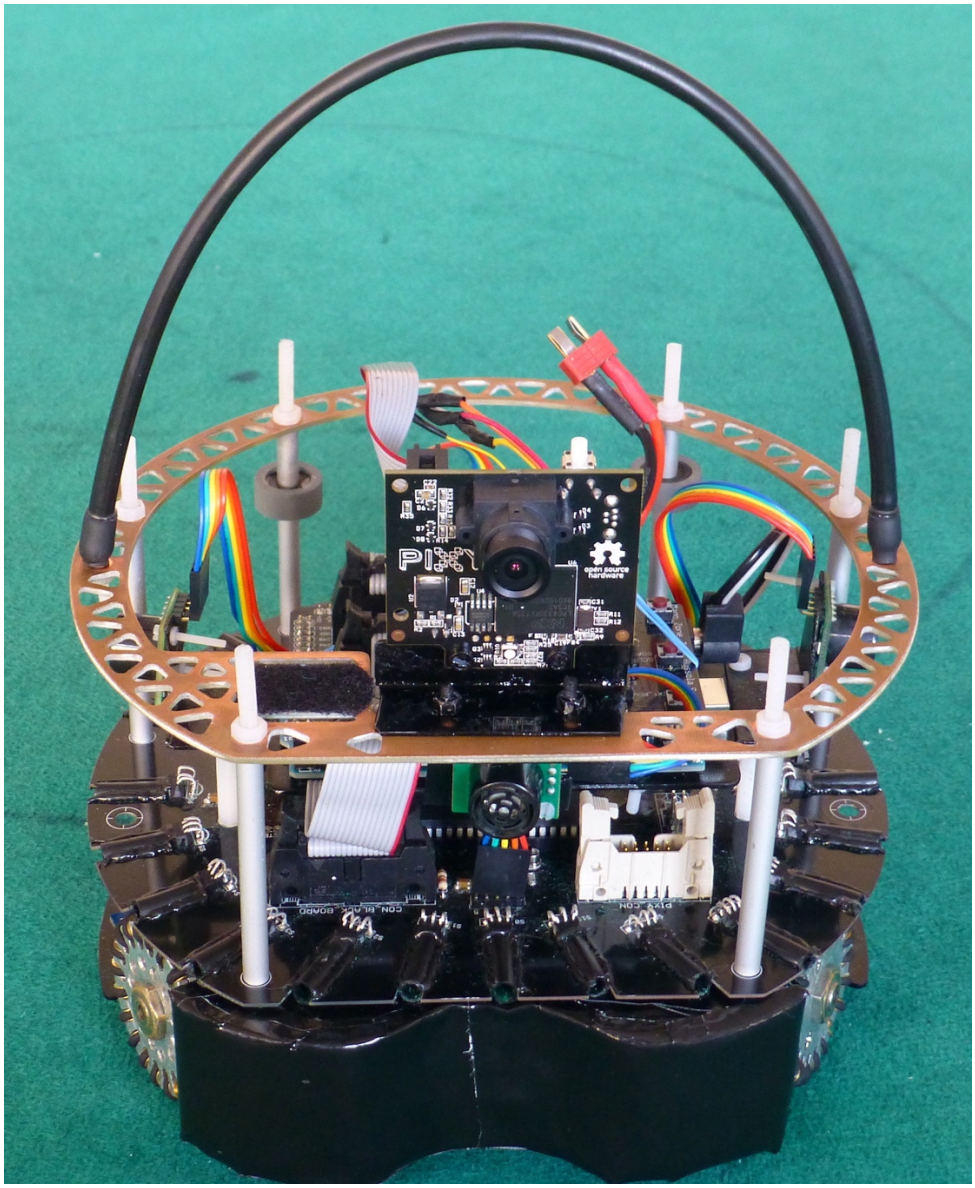
Hobbies: Hacker, Music Composer, Gymnastics

SALVADOR



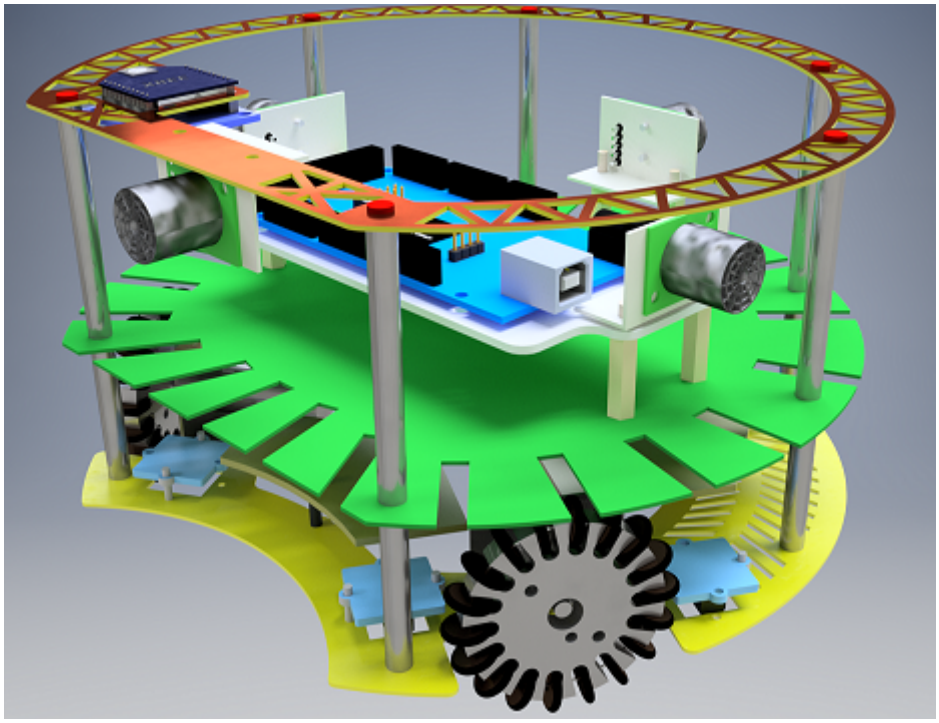
- Age: just born
- Role: goal keeper
- Electronics: New boards, IMU, line sensors
- Mechanics: New CAD design, handle and columns
- Software: New strategy

MANFREUS



- Age: just born
- Role: goalie
- Electronics: New boards, IMU, line sensors, Pixy
- Mechanics: New CAD design, handle and columns
- Software: New strategy

ROBOT DESIGN



Three Electronic Boards:

- Motor Board
- Ball Board
- Arduino Board & Shield

Designed with

AUTOCAD & EAGLE CAD

ROBOT INNOVATION

Each robot is brand new because also our team is completely different: young people enthusiastic and workaholic.

Our major improvements:

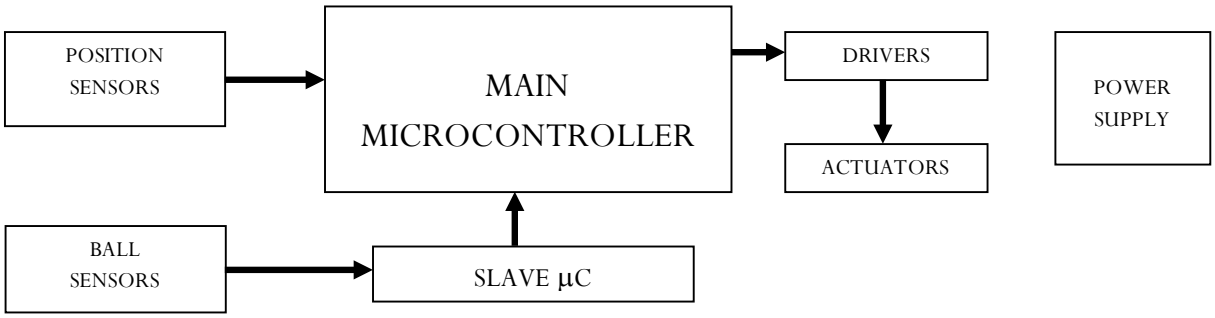
HARDWARE

- 20 ball sensors instead of 16 for better identification of ball position
- Line sensors connected to PCINT input of AVR 2560 for interrupt history tracking
- BNO-055 9-axis motion sensor instead of HMC6352 Sparkfun Compass Module to avoid magnetic interferences
- All boards redesigned and realized as printed circuits to avoid wire mess and false contacts
- All boards designed for further improvements
- AtMega 644p as "brain" of Ball Board instead of PIC1820
- SPI Master Slave communication between the Arduino board and the Ball Board
- Analog /Digital X connection with CMUcam5 Pixy for opponent goal detection
- Homemade design of aluminum columns
- Homemade design of handles
- Homemade design of wheels

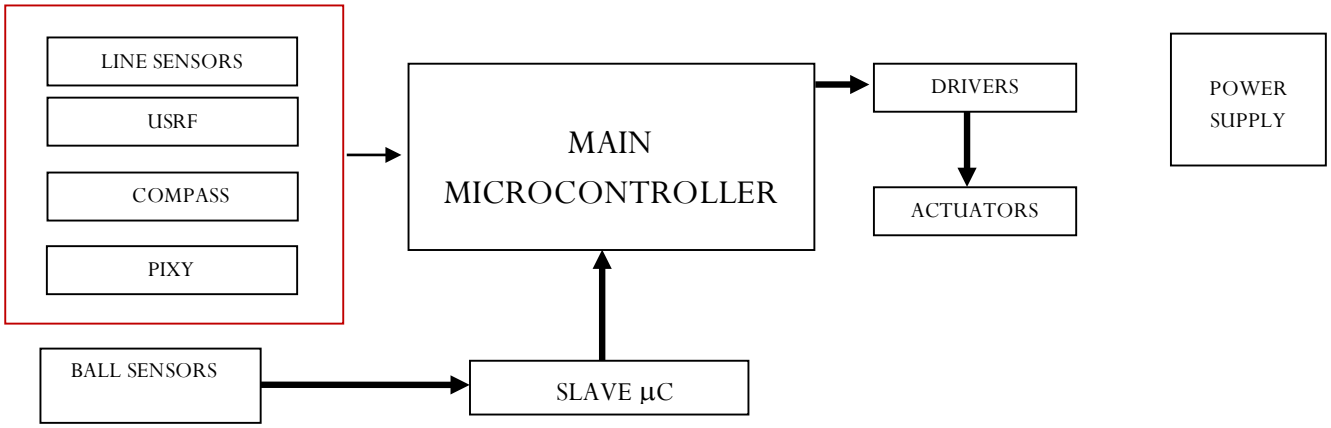
SOFTWARE

- New statistic computation of robot position
- New sensor fusion evaluation including also Pixy camera readings
- Totally redesigned interrupt robot strategy to escape from boundary lines
- Totally redesigned goal keeper strategy
- Totally redesigned forward strategy

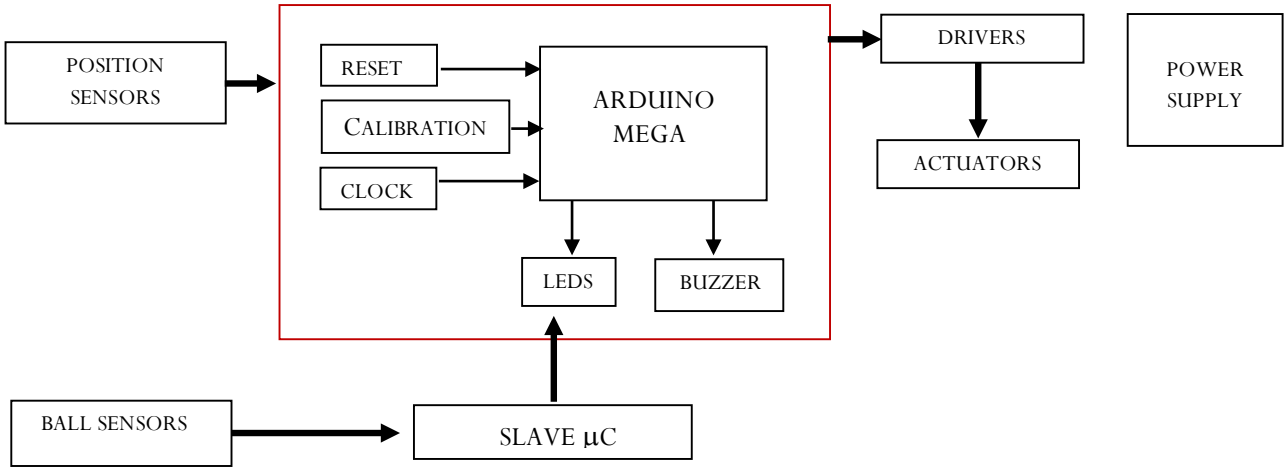
ROBOT BLOCK DIAGRAM

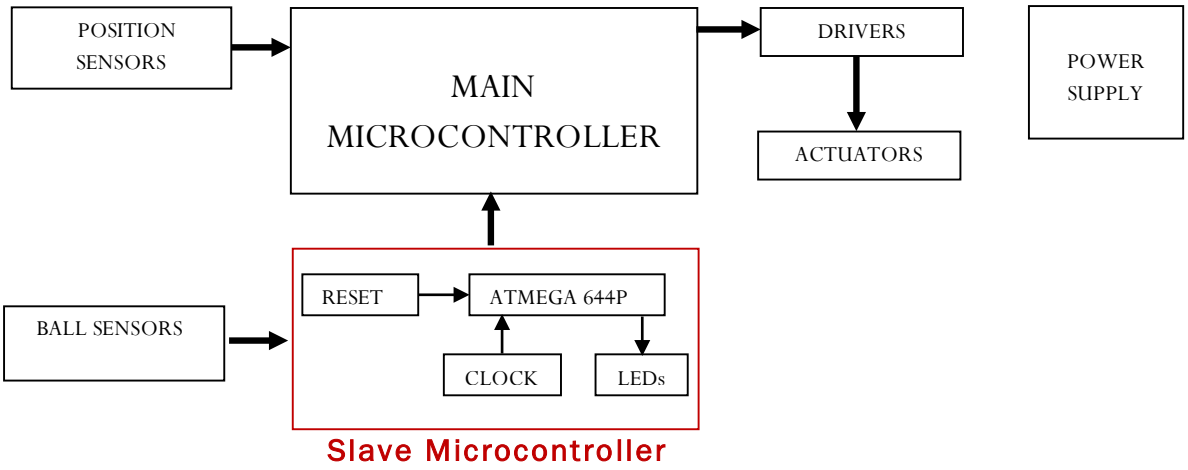


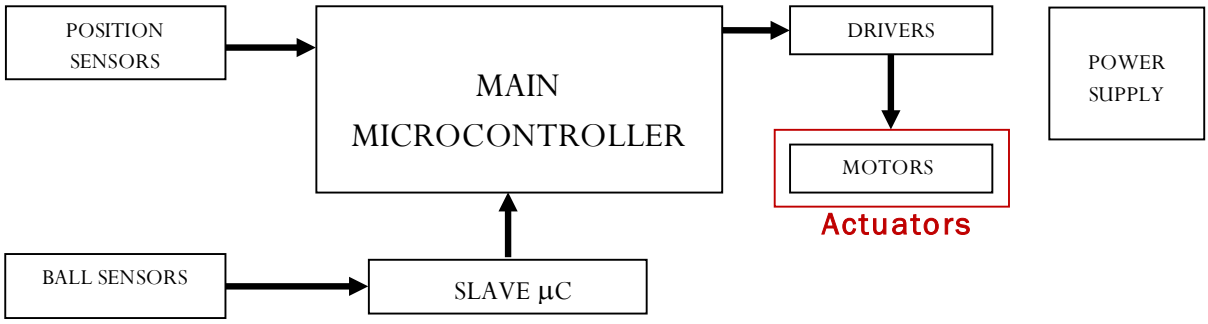
Position sensors

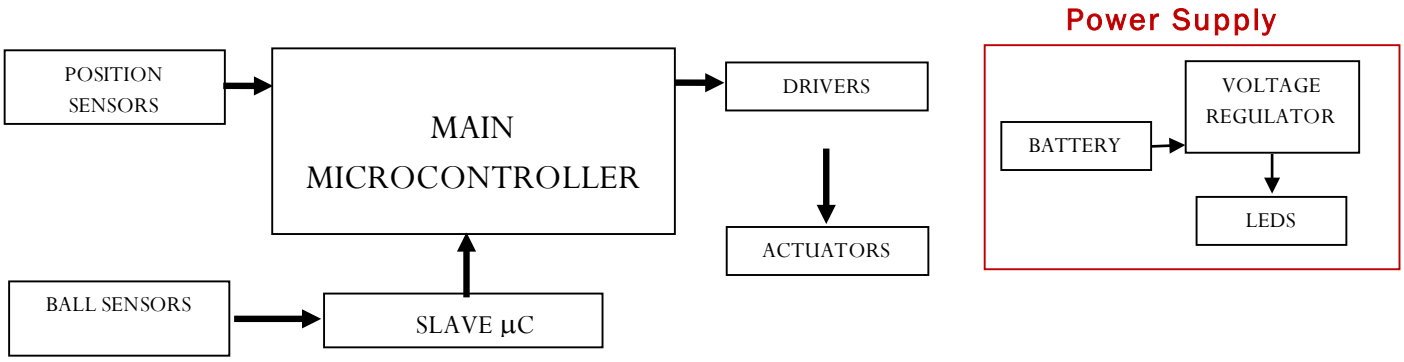


Main Microcontroller




POSITION SENSORS
MAIN MICROCONTROLLER
DRIVERS
ACTUATORS
POWER SUPPLY
BALL SENSORS
RESET
ATMEGA 644P
CLOCK
LEDs
Slave Microcontroller


POSITION SENSORS
MAIN MICROCONTROLLER
DRIVERS
POWER SUPPLY
MOTORS
Actuators
BALL SENSORS
SLAVE µC

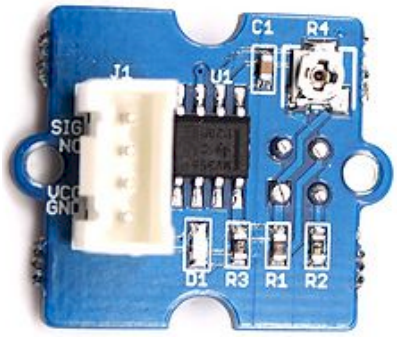

POSITION SENSORS
MAIN MICROCONTROLLER
DRIVERS
ACTUATORS
BALL SENSORS
SLAVE µC
Power Supply
BATTERY
VOLTAGE REGULATOR
LEDS

ROBOT COMPONENTS

POSITION SENSORS

➢ *Line sensors*

GROVE INFRARED



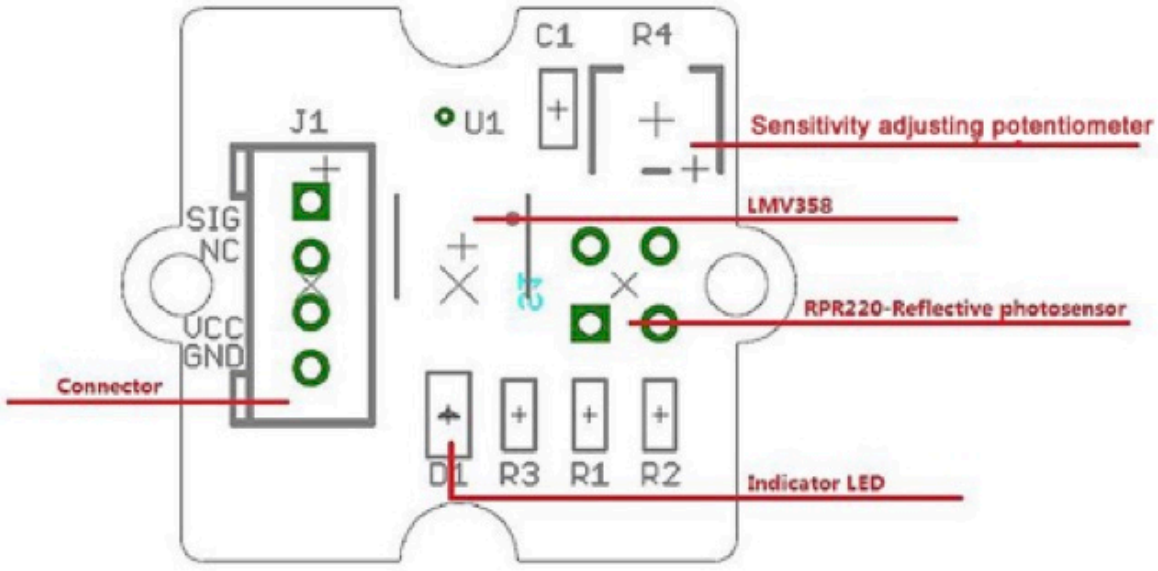
To detect the white lines on the playground and contain our robots within the field bounds, we need line sensors.

Each robot is equipped with six line sensors located on the edge of the lower chassis, each couple just near to the correspondent wheel.

Our team choose the "Grove - Infrared Reflective" sensors by Seeed.

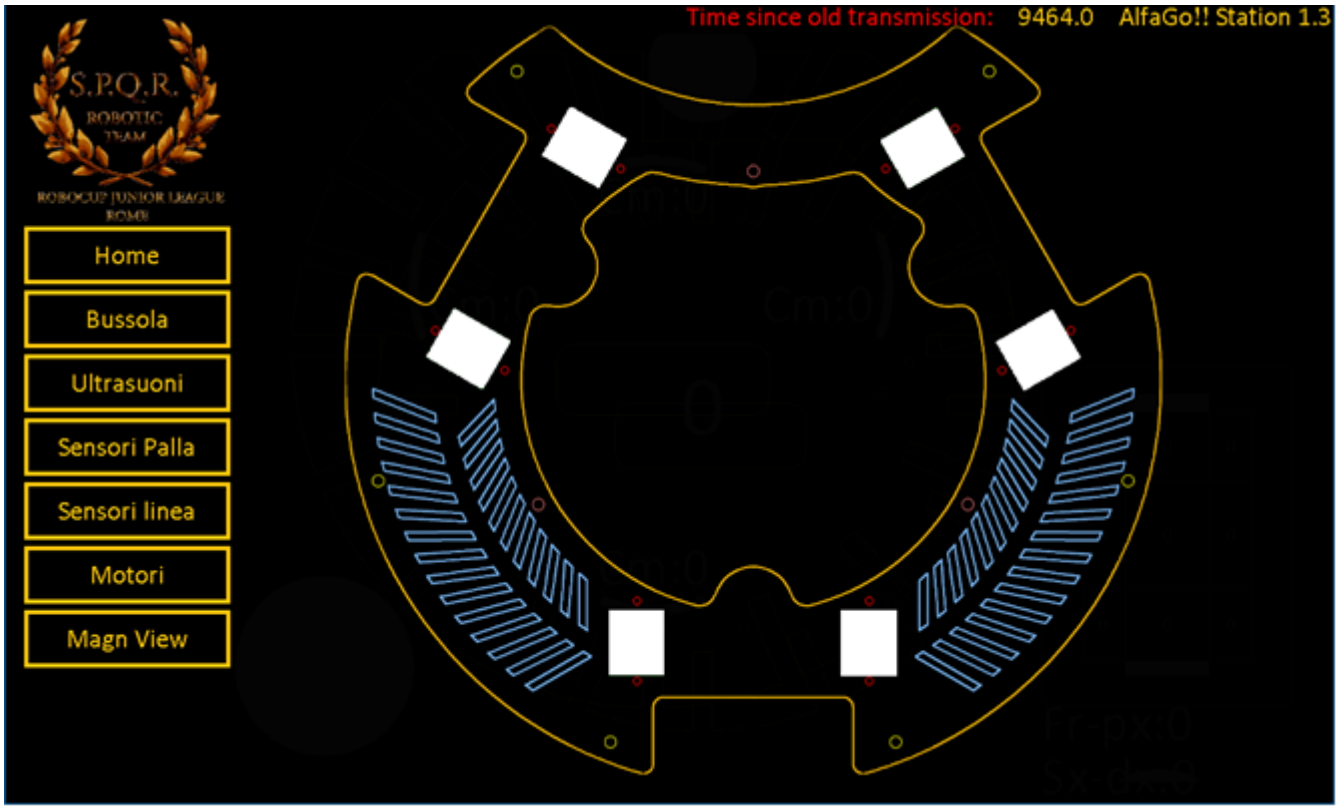
This module utilizes a RPR220 reflective photo-sensor module to detect color and distance. When a light object approaches, the signal intensity received by the infrared reflective sensor increases and the indicator LED on board turns red. When a dark- object approaches, the intensity decreases and the LED turns off.

So when a white line is detected the analog comparator inside the module outputs 0; in any other case the module outputs 1. It is possible to change the sensitivity of the RPR220sensor using the on board trimmer (R4).



The J1 connector links the lower board (line sensors motor and motor drivers) to the main Arduino board

The line sensor outputs are connected to six PCINT (Pin Change INTerrupt) pins of the Arduino Mega board. These pins share the same interrupt routine that manages the robot behavior forcing it to stop and to get out from the line. It is possible to check which sensor has detected the line, if the robot is leaving the dangerous situation and memorize the interrupt "history". **This is one of the main changes in our robot design and software strategy.**



In our debug station line sensors are showed when no line is detected as 6 small black squares on the robot base.

When a sensor detects the line, it turns in to a white square showing the occurrence of the interrupt.

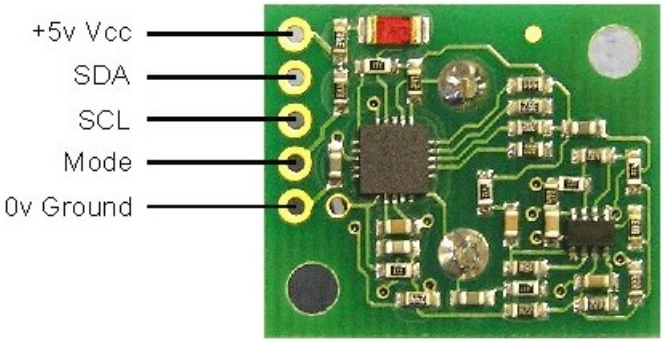
So it is possible to check any malfunctioning and to keep track of the response time of the robot

➢ *Ultrasonic Range Finder*

SRF02

The Ultrasonic Range finder finds objects with sound the same way a bat or a dolphin does. One part of the sensor is a speaker that sends out a sound wave. The other part is a microphone that then measures how long it takes for the sound wave to come back. The longer it takes to come back, the further away the object.

After a long evaluation of the range finders on the market we decided to measure the robot position in the playground using SRF02 Ultrasonic range finder by Devantech connected via I2C protocol to the Arduino board.



Each sensor can be identified with an address chosen in a pool of 16; so it is theoretically possible to connect 16 sensors on a single robot.

Our robots are equipped with 4 sensors pointed to:

Address		Long Flash	Short flashes
Decimal	Hex		
224	E0	1	0
226	E2	1	1
228	E4	1	2
230	E6	1	3
232	E8	1	4
234	EA	1	5
236	EC	1	6
238	EE	1	7
240	F0	1	8
242	F2	1	9
244	F4	1	10
246	F6	1	11
248	F8	1	12
250	FA	1	13
252	FC	1	14
254	FE	1	15

- North 0° Front
- East 90° Right
- South 180° Rear
- West 270° Left

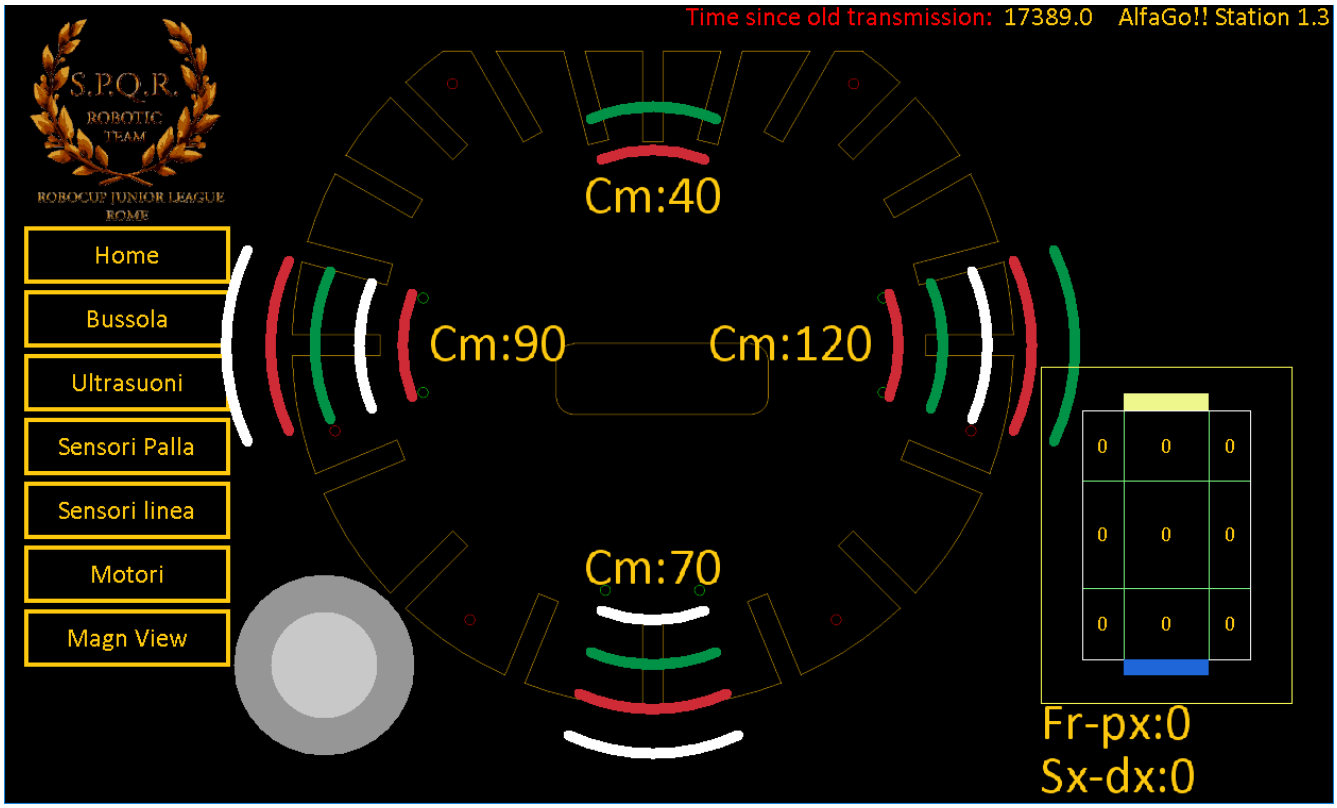
We use the following addresses:

- Nord 224
- Est 226
- Sud 228
- Ovest 230

Our software triggers the start of the ultrasonic beam and the result is available only after 65 ms.

We take note of the sensor status. So in our loop we don't trigger the sensor twice because we know that a measure is in progress and we read the result only when it is ready.

Results are available in inch, centimeters and microseconds: as Italian people, we choose centimeters



Our debug station shows on the screen the sensor position, and the readings in centimeters. Waves of length of increasing dimension show the distance visually.

Using these data our software "guess" the position of the robot on the field.

This one has been ideally divided in 9 areas and the program evaluates in which area the robot is (or should be). A small field in the right corner shows the nine zones: the zone in which the robot supposes to be is shown in white.

Operatin a fusion of all data collected from sensor lines, ultrasonic range finders and line sensors our software chooses the game strategy. In such a way we also can compensate reading errors.

➢ Compass

BNO-055

We choose the Bosch's new BNO-055 9-axis motion sensor in a super-small (10 mm x 10 mm), wearable size. The BNO-055 has an embedded Cortex M0 ARM processor as well as accelerometer/gyro and magnetometer for a purely hardware absolute orientation solution.

In effect the new trend in motion sensors is to embed powerful processors with sensors to allow direct calculation, avoiding the need for users to program their own sensor fusion or to take up valuable microprocessor memory or processing power in crunching the numbers for sensor fusion.

This device is composed by a 9-axis motion sensor (the BMX-055) coupled with a Cortex M0 ARM processor to perform the 9-axis sensor fusion. No external magnetometer and no microcontroller processing is required; again the quaternions, linear acceleration, gravity vector, and heading information are directly readable from the BNO-055 registers. This is a compact and powerful motion sensing solution that makes absolute orientation and sophisticated motion control available also for an Arduino board.

This small-form-factor board is hardwired for I2C communication with 4K7 pull-up resistors on the board.

The hardware sensor fusion is updated at a fixed frequency = 100 Hz i.e. measures are performed every 10 ms.

Our software programs the BNO-055 in Mode IMU-PLUS

Our debug station monitors the compass too. A specific screen displays:

- the value (degrees) indicated by the compass
- the status of the switches used for the calibration and the regulation
- the degrees between the robot and the opponent goal

All data are registered for further analysis and historic data can be shown.

Obviously the North shown in the picture is not the real one.

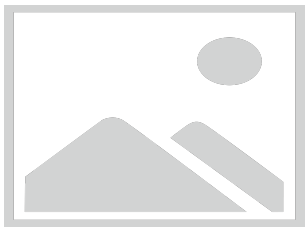
At the beginning of each match and when teams swap fields we have to inform the software about the position of the opponent goal.

First of all, we have to calibrate the compass moving the robot around for a while as it is suggested in the compass data sheet. Then when the robot is facing the opponent goal we press the CAL button. The software stores into the EEPROM the compass reading i.e. the orientation of the opponent goal versus the geographic North.

When Arduino starts up, it reads the data stored and every reading from the compass during the game is corrected by this factor, making the opponent goal our “Virtual North”.

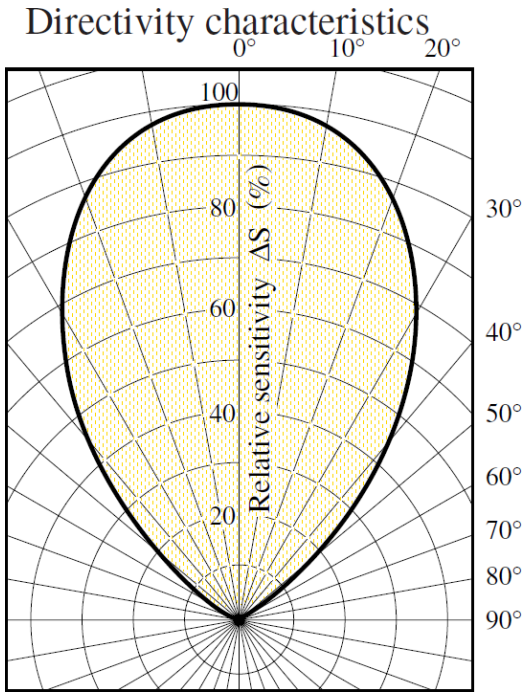
➢ Ball Sensors

TSSP4038



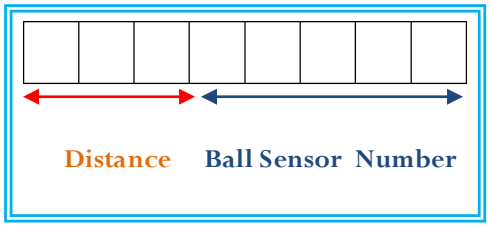
To find the IR ball we need ball sensors. Our robots are equipped with twenty "TSSP4038", a photo IC infrared 38.0 KHZ sensor produced by Panasonic Electronic Components shown in the picture.

Pin 1 = OUT
2 = GND
3 = Vs



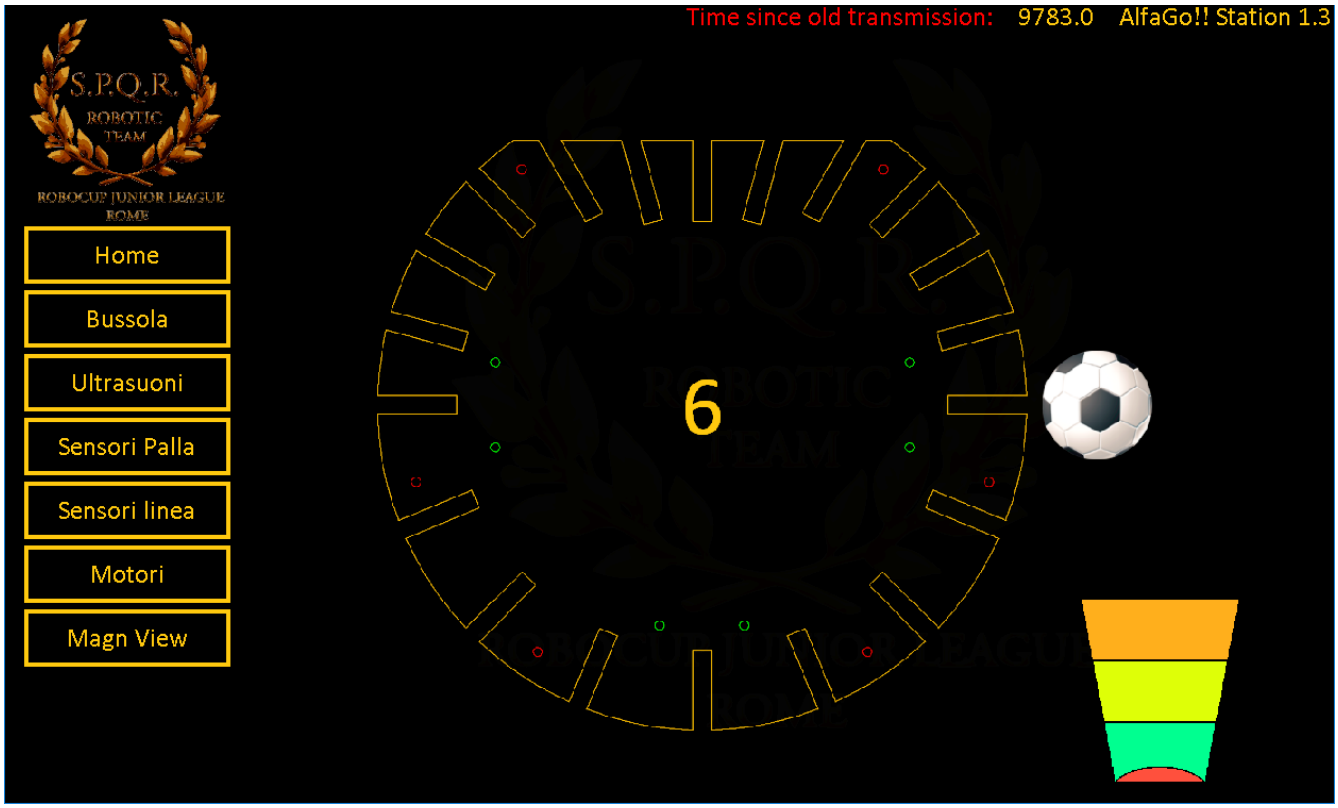
The main problem is the wide range of the TSSP4038 beam detection (see the picture).
It can detect also IR signals at 50° from the sensor axis. To make our sensors more directional we have inserted at the end of each one a small black plastic pipe. So the sensor can "see" only just in front of it reducing the spurious signals to almost zero.
This trick makes the sensors range very sharp and the microcontroller is able to identify quite easy the "best" sensor that is facing the ball.

On our ball board the AtMega644p microcontroller collects data from IR sensors in loops of 9 ms, so we can track 3 period of the pulsed IR emitted from the ball. When the Master Mega Board needs the readings from the AtMega644p, it triggers an interrupt via SPI and 1 byte is exchanged.
Bits 0…4 are the number of the sensor that is in front of the ball, meaning that has collected the maximum readings



Bits 5…8 evaluate the distance of the ball because from the value of the maximum we can guess the distance.

Our debug station has a specific screen that shows all data collected by the “ball board”.



Putting the ball in front of each sensor, the debug station shows graphically the “best” seeing sensor drawing a small football ball in front of it. The number of the sensor is shown in the center of the “robot”.

The estimated value of the distance is represented by the length and color of the bar on the right side:

green (0-15 cm),
yellow (15-25 cm)
orange (25-300 cm).

The small red spot at the bottom of the bar symbolizes the sensor.

➢ *Robot vision*

If a robot to has to perform a task such as picking up an object, chasing a ball, and you want a single sensor to help accomplish all of these tasks, then a vision sensor is needed.

Vision (image) sensors are useful because they are flexible. With the right algorithm, an image sensor can sense or detect practically anything. But there are two drawbacks with image sensors: 1) they output lots of data, dozens of megabytes per second, and 2) processing this amount of data can overwhelm many processors. And if the processor can keep up with the data, much of its processing power won't be available for other tasks. Being our robot equipped with Arduino mega microcontroller, we decided to take a limited information from our vision sensor: the PIXY camera

PIXY is used to detect the opponent goal that is blue or yellow because we must switch field in the middle of the game.

Pixy is unique because you can physically teach it what you are interested in sensing.

PixyMon is an application that runs on Windows, MacOs and Linux. It allows you to see what Pixy sees, either as raw or processed video. It also allows you to configure the Pixy, set the output port and manage color signatures. PixyMon communicates with Pixy over a standard mini USB cable. Pixy can learn seven color signatures, numbered 1-7.

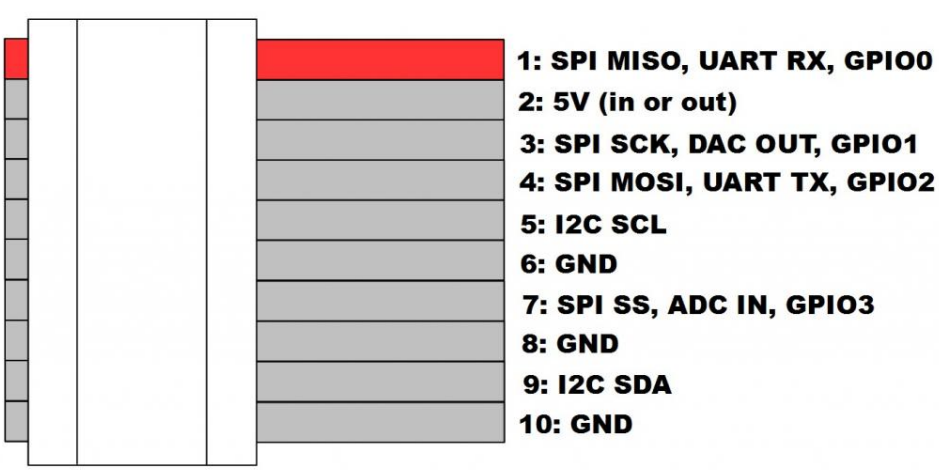
To determine how you want to talk to Pixy configuring the output interface through PixyMon

We choose **analog/digital x** (Mode 4) This will output the x value of the largest detected object as an analog value between 0 and 3.3V (pin 3 DAC OUT). It also outputs whether an object is detected or not as a digital signal (pin 1 of the I/O connector GPIO0).

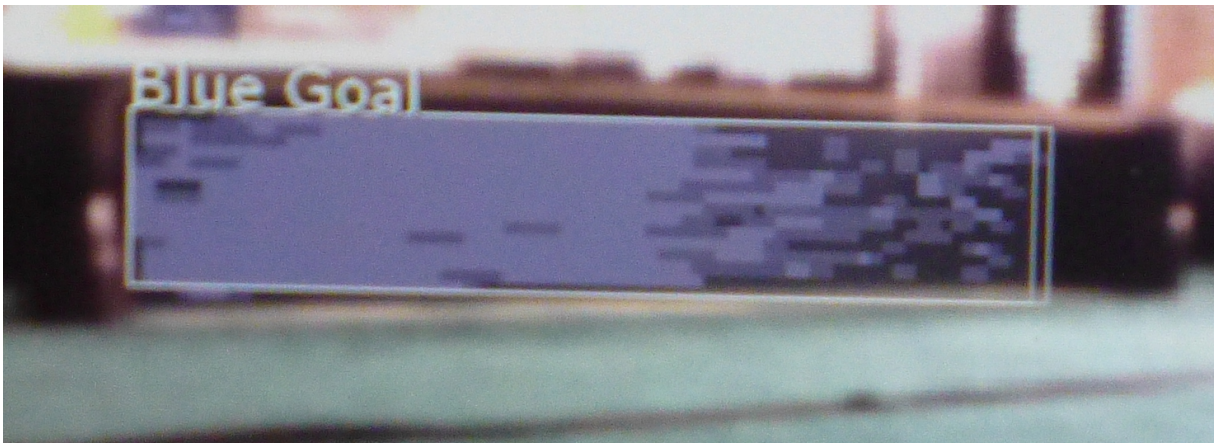
In this way our software knows:

- Reading pin 1 if the opponent goal is in sight
- Reading pin 3 if the robot is centered respect the opponent goal

The PIXY connector is shown in the picture

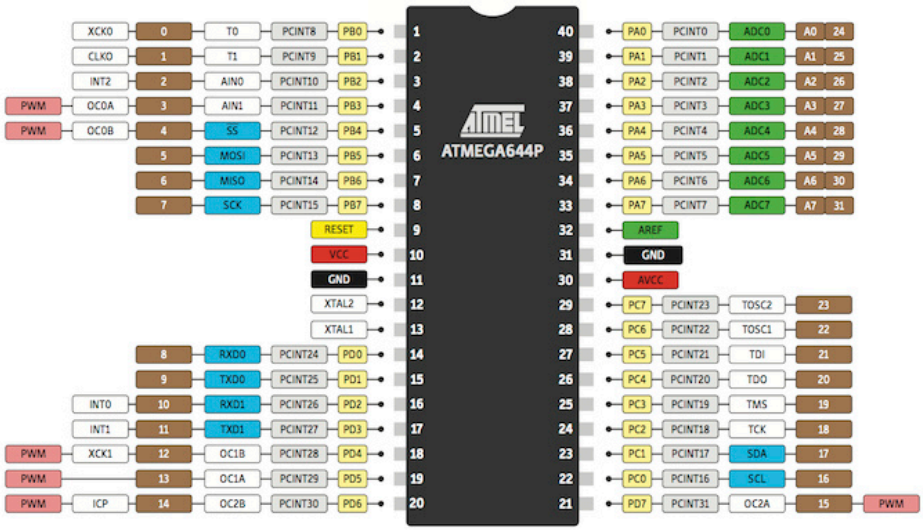


The following pictures show the opponent goals "seen" by the PIXY



➢ *Slave Processor*

ATMEGA 644P



Located on the Ball Board, the AtMega 644p main task is to check continuously the ball sensors and transmit to the Mega Board the information required via SPI.

The ball sensors are connected to PINC, PINA, PIND input ports of the microcontroller.

The program for the ATMega 644p has been written in Arduino IDE. We download the code via the USBASP programmer.

This one is a cheap programmer and eliminates the need for a bootloader which occupies 2KB in the on-chip flash.

Power Supply

➢ Power Supply

LIPO BATTERY

The power main source is a 3 cells 12V LIPO battery.
Motors are connected to the battery via relative drivers.
The battery is connected also to 8V and 5V voltage regulators to drive correctly all electronic components.

➢ Voltage regulator

VOLTAGE REGULATOR

On the ball board there are two 78XX voltage regulators used to reduce and stabilize the 12V:

One provides a 5V voltage to power up many components of the robot e.g. sensors and AtMega644p. In the previous versions of the robot the 12V voltage was connected directly to the Arduino Mega power input jack. This solution gave some problems because the internal voltage regulator of the Arduino was stressed and overheated. To solve this problem we put another voltage regulator to provide a 8V stabilized voltage and this power output is now used to power up the Arduino Mega. This voltage level reduction helps the internal voltage stabilizer of the Arduino reducing in this way the thermal stress of the component.

Both the voltage regulators belong to the 78XX family and have a TO-220 package that allows a better heat dissipation. This package has 3 pins that are:

1-Input (voltage to be stabilized)
2-GND
3-Output (stabilized voltage)
As suggested in the component datasheet, some capacitors have been added in order to obtain better stabilization and to better compensate the voltage shifts.

➢ Clock

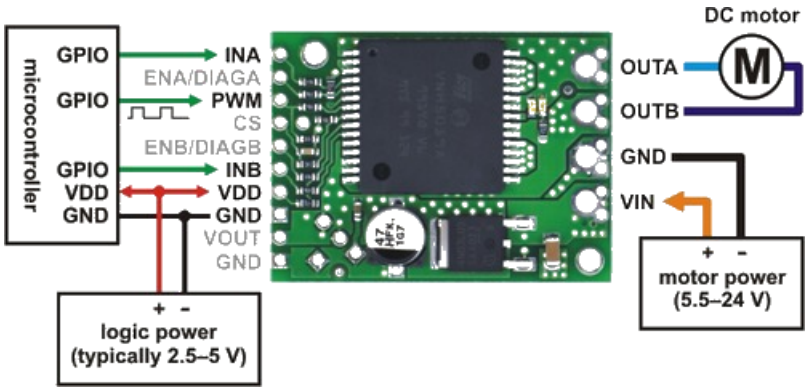
CLOCK

Only one component on the robot requires an external clock: the AtMega644p. For this reason, we added an oscillator that is a small 16 MHz quartz crystal with the HC-49/US package. It is connected to the XTAL1 e XTAL2 pins of the AtMega644p and to avoid interferences two capacitors are also connected to the same component.

➢ *Motor Driver*

VNH5019A

To drive each motor our robots have on board a Pololu module the VNH5019A, a compact breakout board for ST's high-power VNH5019a fully integrated H-bridge motor driver

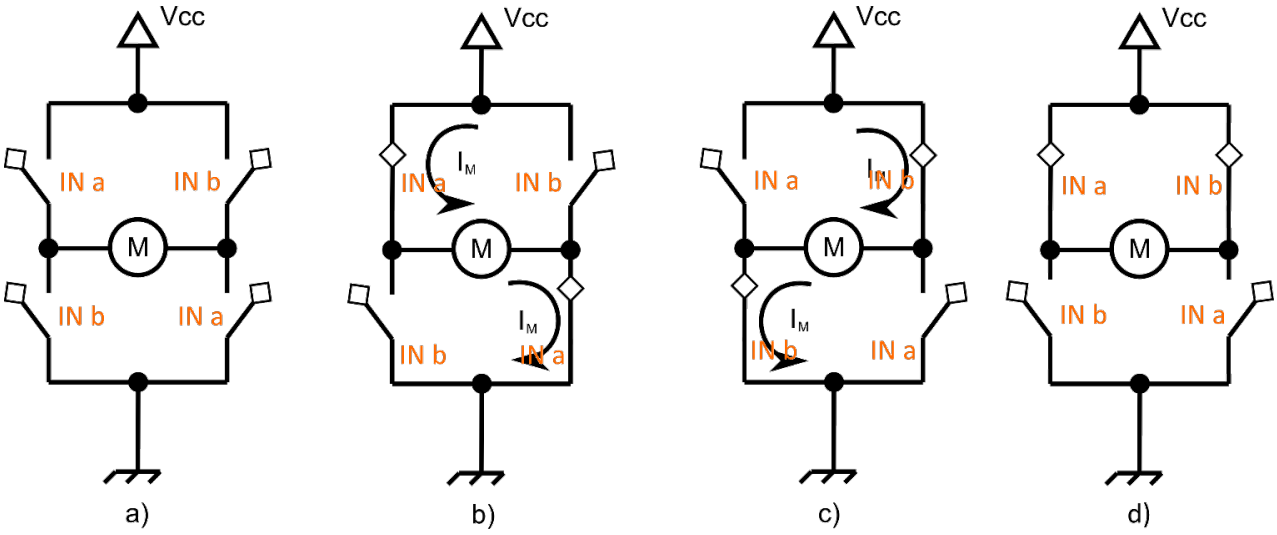


Our robot has 3 motors, so our "motor board" has 3 motor drivers on it.

Type	$R_{DS(on)}$	I_{out}	V_{ccmax}
VNH5019A-E	18 mΩ typ (per leg)	30 A	41 V

This IC operates from 5.5 to 24 V and can deliver a continuous 12 A (30 A peak).

The H-bridge is used for bidirectional speed control of a single brushed DC motor. It is composed by 4 transistor driven by 2 pins , INA INB , that control the direction of motor rotation. As shown on the picture the current flow depends on INA and INB



The following table shows the influence of INA INB status on the operative mode, the pin OUTA (Output of half-bridge A connected to one terminal of the DC motor). and the pin OUTB (Output of half-bridge B connected the other terminal of the DC motor).

Case	IN_A	IN_B	OUT_A	OUT_B	CS (V_{CSD} = 0 V)	Operating mode
d)	1	1	H	H	High imp.	Brake to V_{CC}
c)	1	0	H	L	$I_{SENSE} = I_{OUT}/K$	Clockwise (CW)
b)	0	1	L	H	$I_{SENSE} = I_{OUT}/K$	Counterclockwise (CCW)
a)	0	0	L	L	High imp.	Brake to GND

The CS pin is the Current sense output. If connected to a resistive load it is possible to check the wheel overload. This pin is not yet implemented In our robots.

The module needs 2 different power supply sources:

- VIN from the battery (12 V in our robot) for the motors
- VDD (5V) for the digital components

Actuators

➢ *Motors*

POLOLU

We opted for 20.4:1 Metal Gearmotor 25Dx50L mm HP 6V without encoder.

www.pololu.com

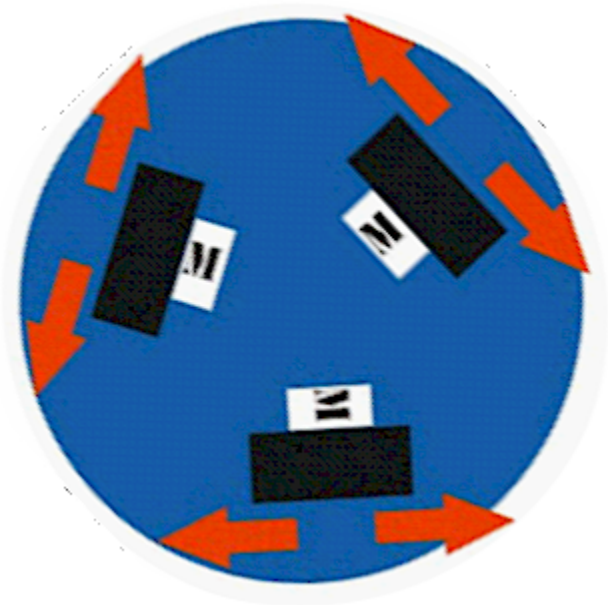
These cylindrical brushed DC gearmotors are available in a wide range of gear ratios and with five different motors (two power levels of 6V motors and three power levels of 12V motors). The gearmotors all have the same 25 mm diameter case and 4 mm diameter gearbox output shaft, so it is generally easy to swap one version for another if our design requirements .

Stalling or overloading gearmotors can greatly decrease their lifetimes and even result in immediate damage. For these gearboxes, the recommended upper limit for instantaneous torque is 200 oz-in (15 kg-cm. Stalls can also result in rapid (potentially on the order of seconds) thermal damage to the motor windings and brushes, especially for the versions that use high-power (HP) motors; a general recommendation for brushed DC motor operation is 25% or less of the stall current.

Holonomic Drive System

Non-holonomic robots are ones that cannot instantaneously move in any direction, such as a car. This type of robot has to perform a set of motions to change heading.

For example, to move a car sideways, complex 'parallel parking' motion is needed. This type of robot can move in both the X and Y direction, but requires complex motions to achieve the X direction.



A robot that uses omni-wheels can move in any direction, at any angle. Holonomic wheels have to be able to roll in two dimensions providing traction in the direction normal to the motor axis and parallel to the floor

A four wheels holonomic robot is easier to design because all the math gets really simpler but three wheels is cheaper and the robot is obviously lighter.

So, our holonomic drive uses three fixed wheels placed 120 degrees apart.

The simplest way to build it is by creating an equilateral triangle. Each motor should by placed on the middle of the triangle side.

The wheels can be independently controlled and the robot can move in any direction and rotate 360° inside its own wheelbase. Each wheel can move the robot forward, but since they are located on the periphery of the robot, they can also rotate the robot's frame

For an omni-wheel robot to translate at a particular angle, each wheel must rotate at a particular rotational velocity and direction. Since the robot is moving at angles, the software has to do trigonometric calculations to determine these wheel speeds.

In our software we have the "**HOLON**" function, which takes as input speed, direction, and rotation.

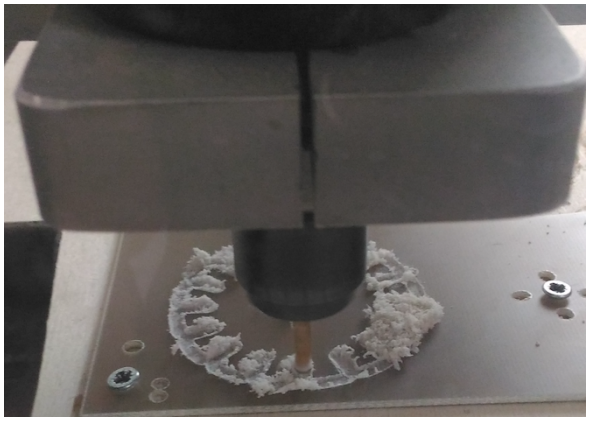
This algorithm calculates the speed to drive each wheel to achieve the desired vector. In addition, if we desire rotation, it adjusts the speed of each motor to achieve the given rotation maintaining the same velocity vector.

WHEELS

Because of our drive system we need special wheels able to be driven with full force but also to slide laterally with great ease. Our wheels have small discs around the circumference which are perpendicular to the turning direction. so they rolls freely in two directions using the rollers mounted around its circumference We employ these wheels because. These holonomic wheels are also known as omni-wheels
There are a lot of omni-weels available on the market

We need robust wheels (i.e. not in plastic). Because of the carpet on the playground, we need also ground traction and a firm grip on the ground otherwise the wheel could slip. This means that discs on each wheel must be made of rubber and their number must be higher than usual.

OMNI WHEELS, MAKING OF…..



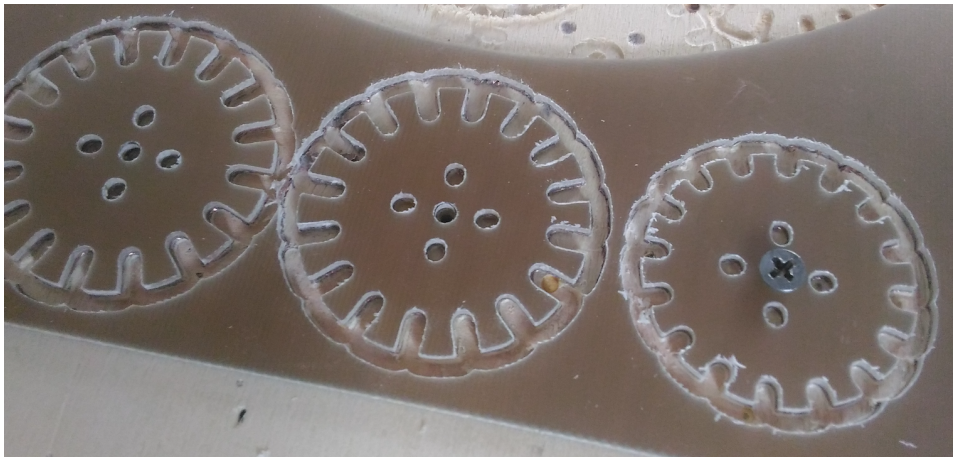
Our wheels are composed by many pieces. This is their production process:

Step 1: Draw CAD for the wheels

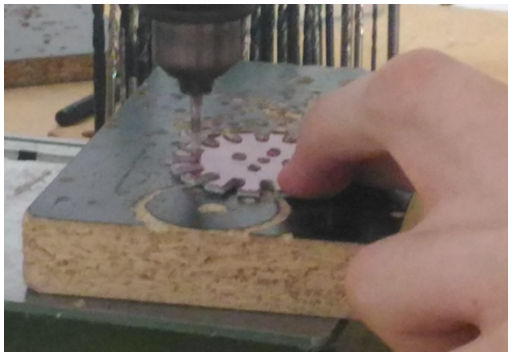
Step 2: Export CAD for CNC

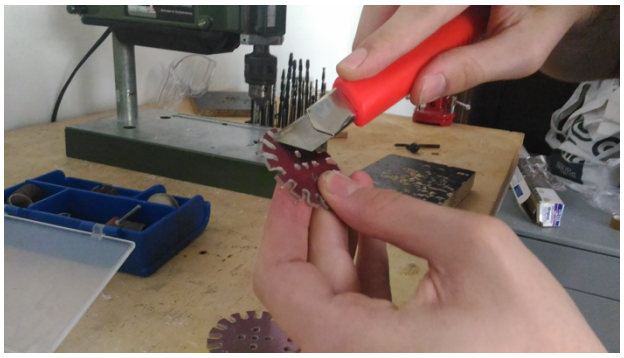
Step 3: Fix the material to be cut

Step 4: Wait the CNC to end



Step 5: Rinse the holes



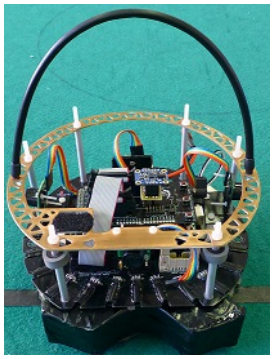


Step 6: Assemble

We are continuously improving our design and hope to have them on our robots as soon as possible

ROBOT ASSEMBLING

When a malfunctioning occurs, it is very important to be able to disassemble the robot and, after a quickly repair, assemble it again easily.



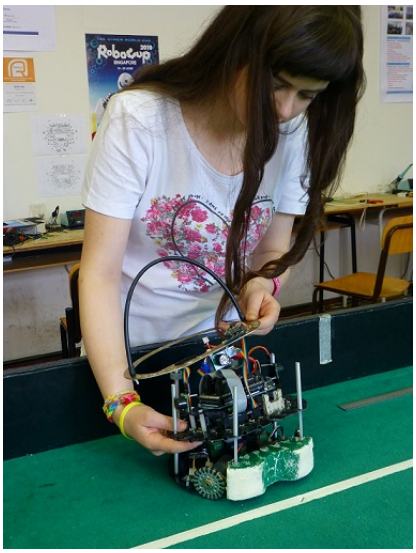
Our group designed a completely new way of assembling, gaining in the process also lightness.

The robot is composed by 3 layers. Between the layers there are aluminium columns with a hole inside.

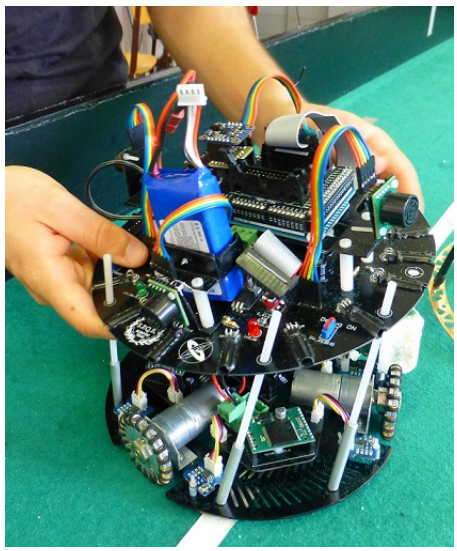
A nylon screw, as high as the robot is, runs inside the columns and is put under tension by a nut on the top.

So, to disassemble the robot you have to unscrew only six nuts.

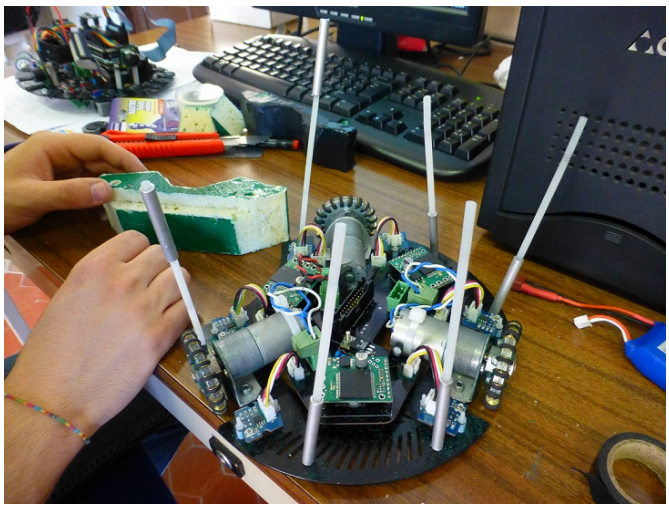
Step 1: Take out the hanle



Step 2: Take out the layer



Step3: Full access to the lowest layer

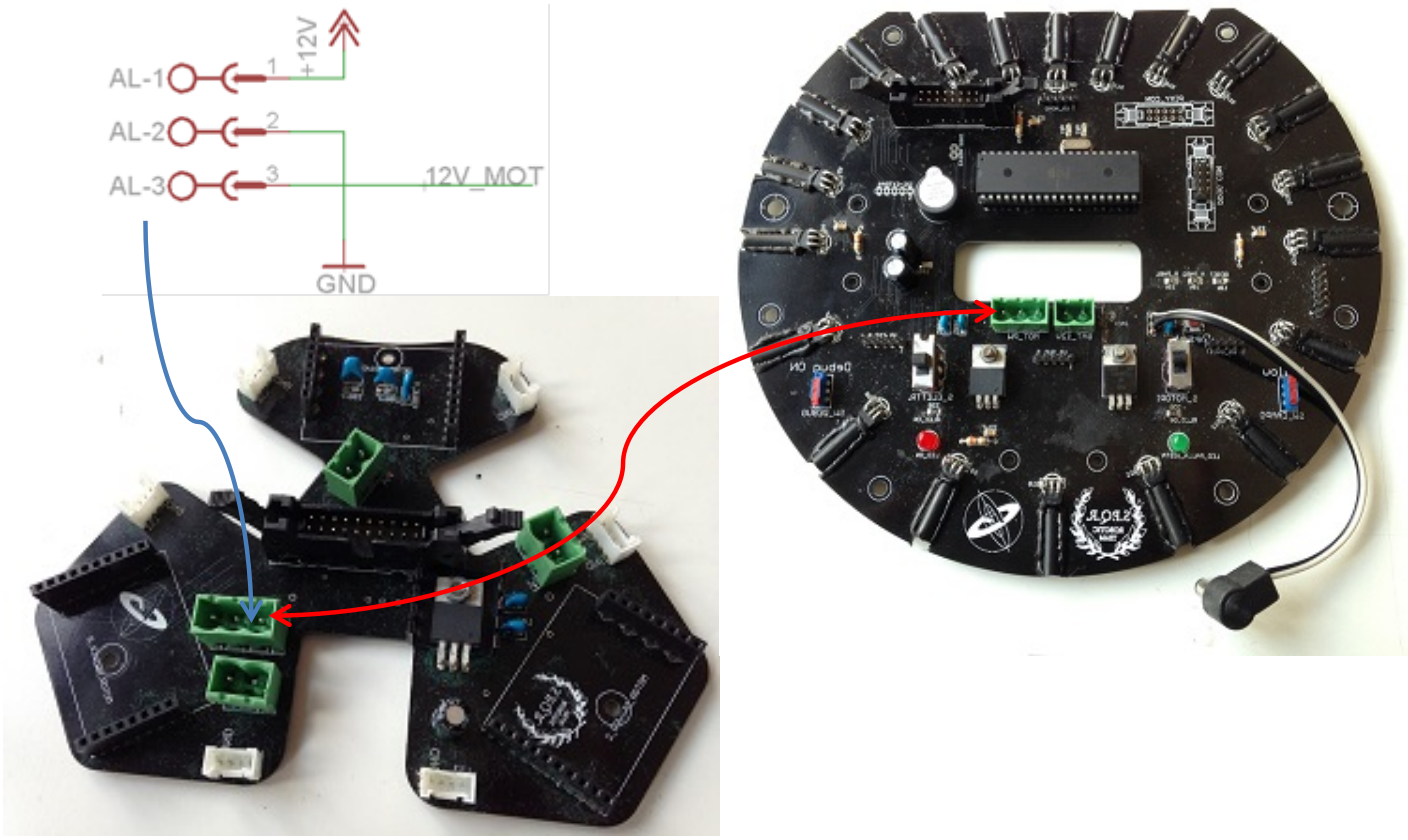


ROBOT BOARDS

➢ Motor Board

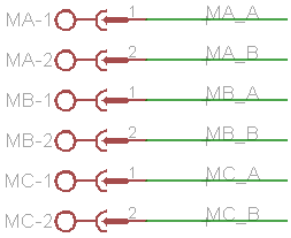
AL: Three pins connector package MSTBV3

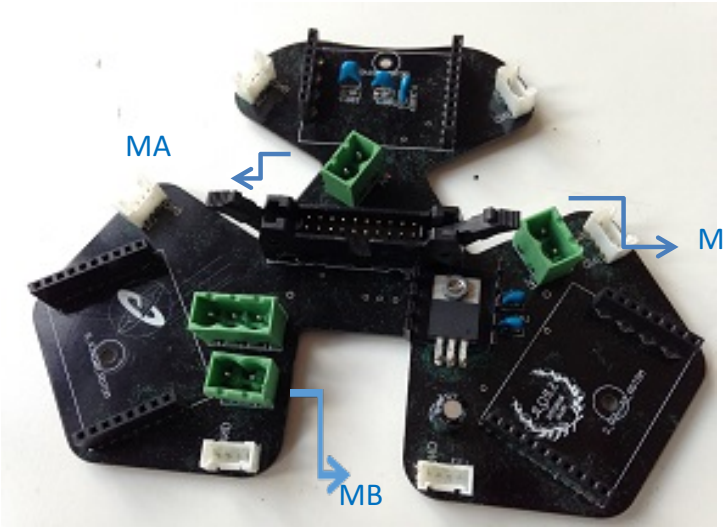
Power supply connector



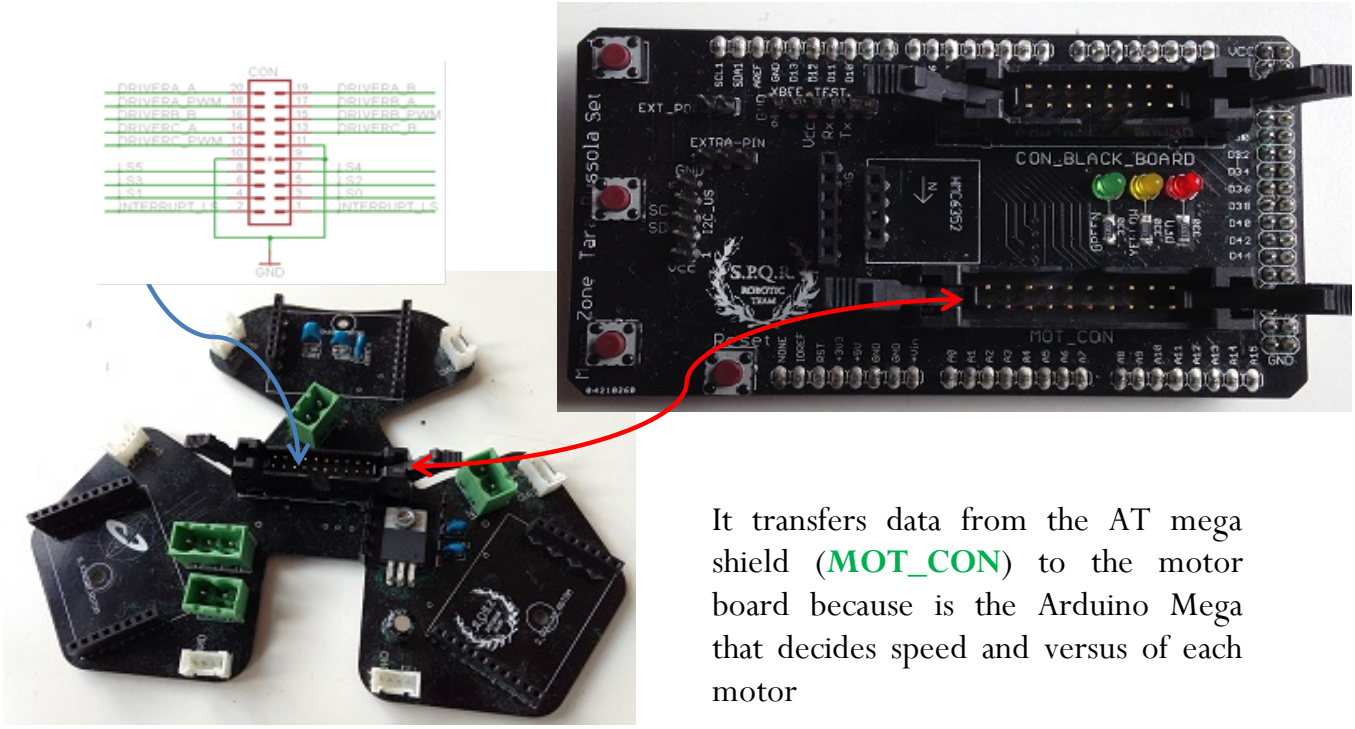
The companion connector is on the centre of the ball board in front of the hole so that the connection between the two boards is quite easy to realize.

MA, MB, and MC: Two pin connectors package MSTBV2.
Their purpose is to connect each motor (MotorA, MotorB, MotorC) to the board.
Their position on the board is near each motors, on the corner of the board.





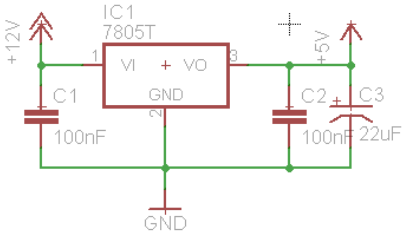
CON: double connector 10 +10 pin



It transfers data from the AT mega shield (**MOT_CON**) to the motor board because is the Arduino Mega that decides speed and versus of each motor

It also transfers to the Arduino Mega the signals from the line sensors that are located on the motor board.
All cables from this connector go to the mega shield thorough the hole in the center of the Ball Board.

IC1: 5V output 7805 voltage regulator. (Package TO220H)



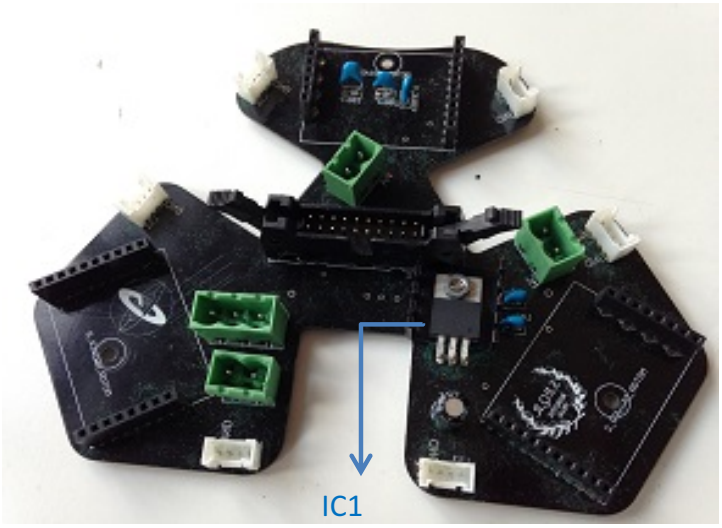
To improve stability capacitors have been inserted:

```
C1   100nF      C-EU025-030X050 C025-030X050
C2   100nF      C-EU025-030X050 C025-030X050
C3    22uF      CAP_POLPTH1
```

It has been strategically positioned on the board because it overheats so:

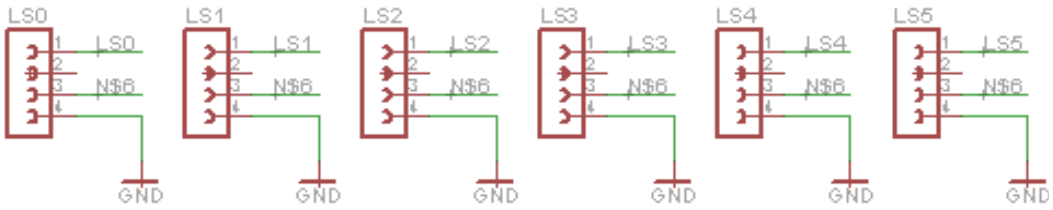
- No neighbouring traces,
- No neighbouring components

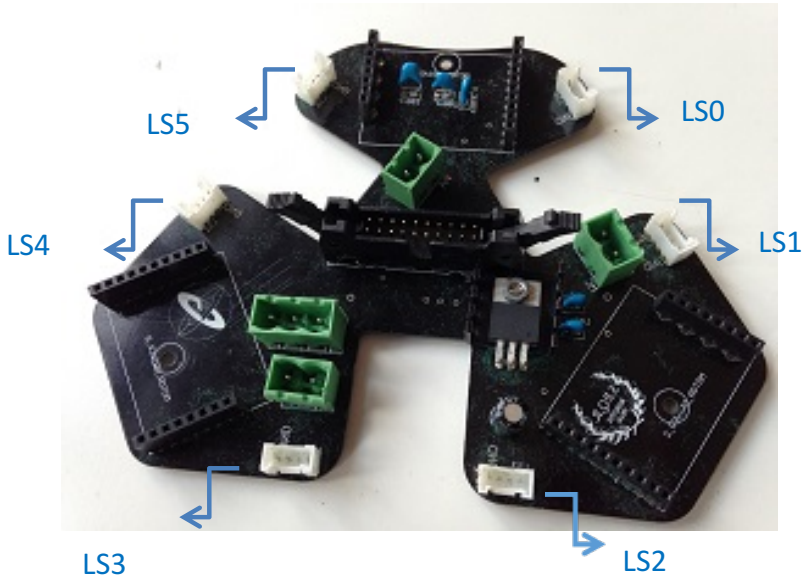
We fix the regulator to the board with a screw in order to increase heat dissipation.



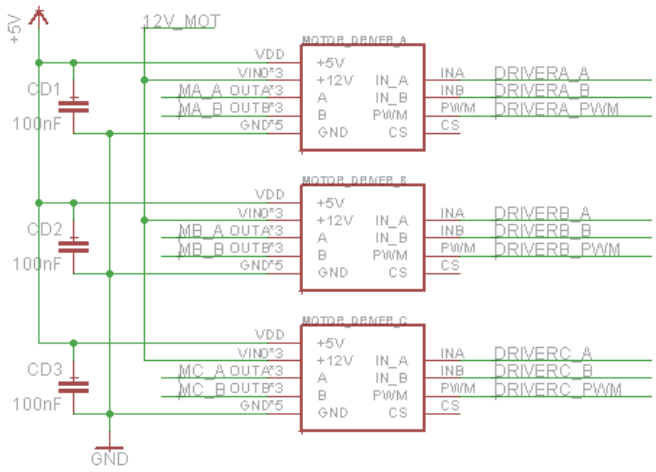
LS0, LS1, LS2, LS3, LS4, LS5:

Groove male connectors (package SPB4) to link the line sensors to the motor board.





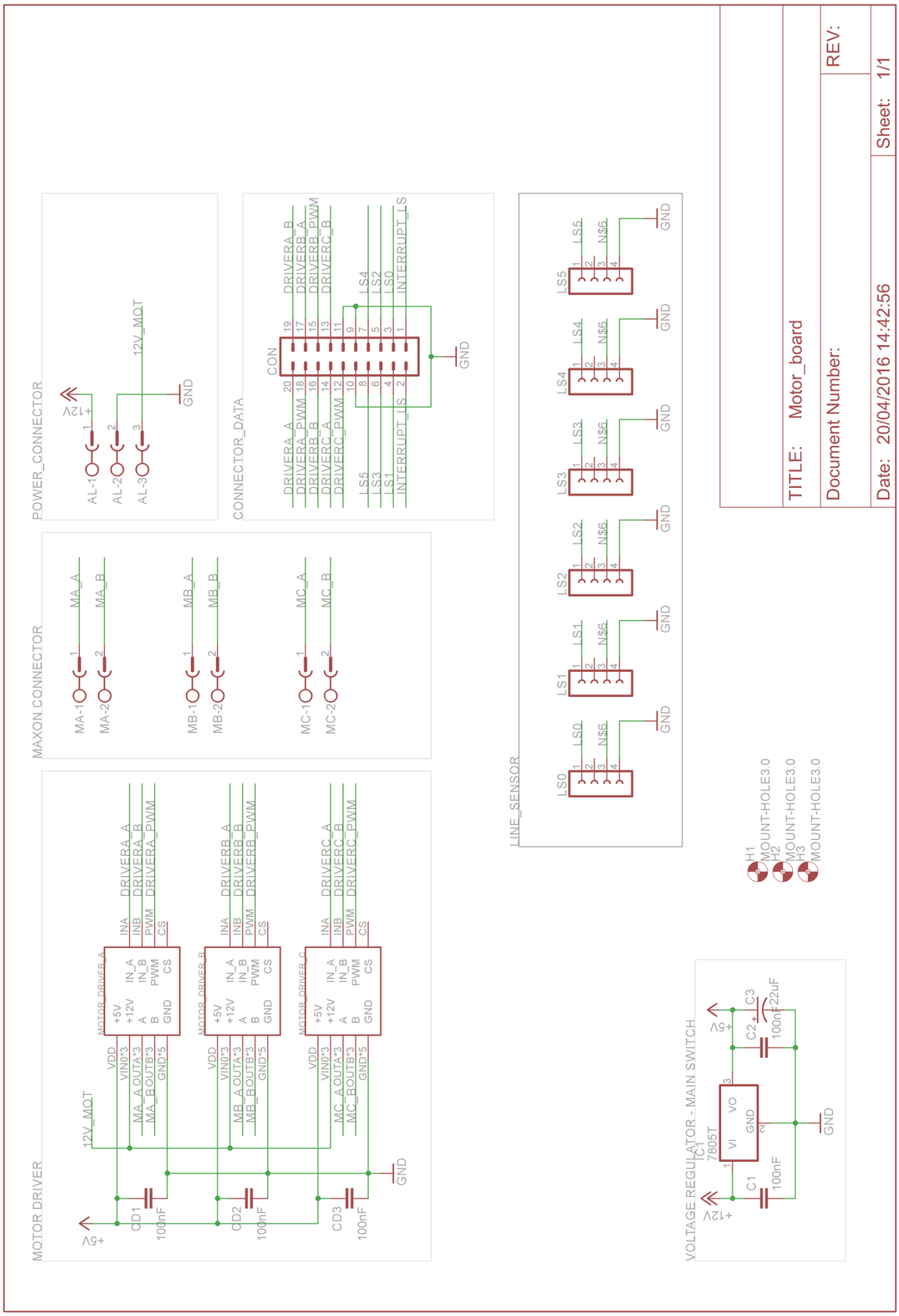
MOTOR_DRIVER_A, MOTOR_DRIVER_B, MOTOR_DRIVER_C:



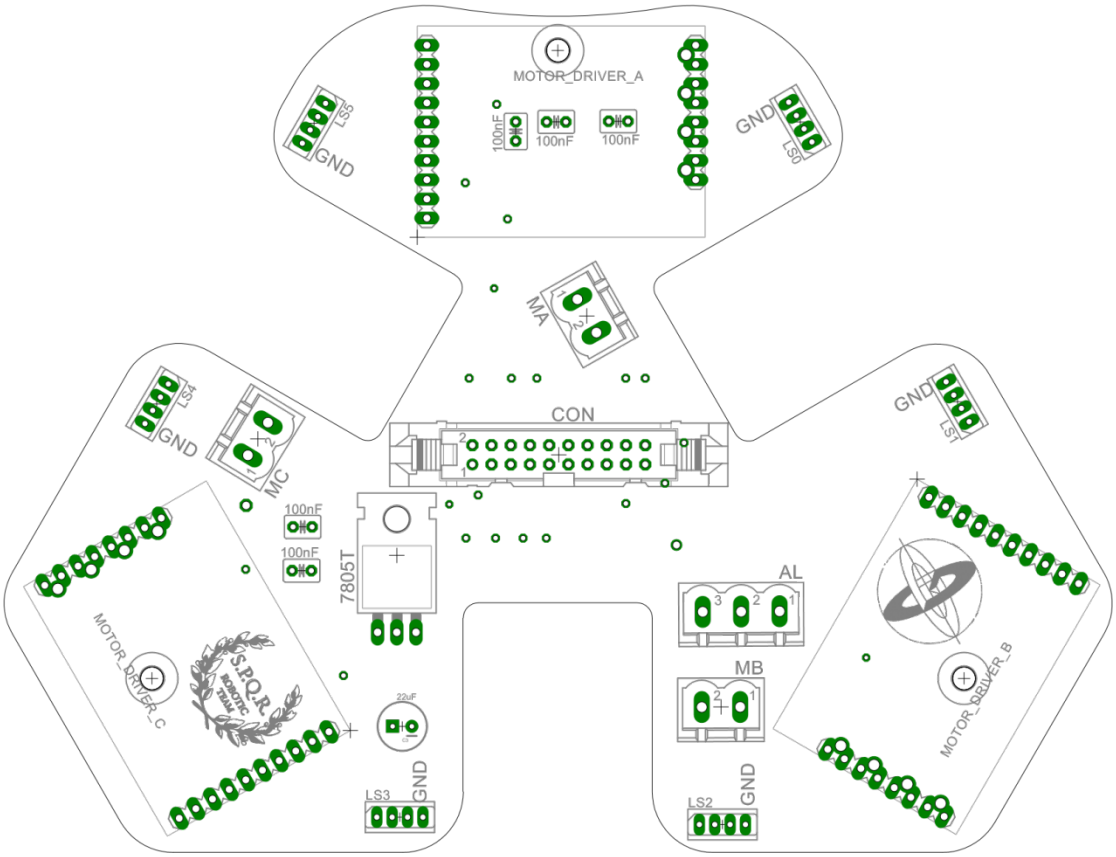
Three VNH5019 drivers for the motors.

Between GND and VDD a 100nF capacitor has been inserted (package C025-030X050).

Motor Board Schematic

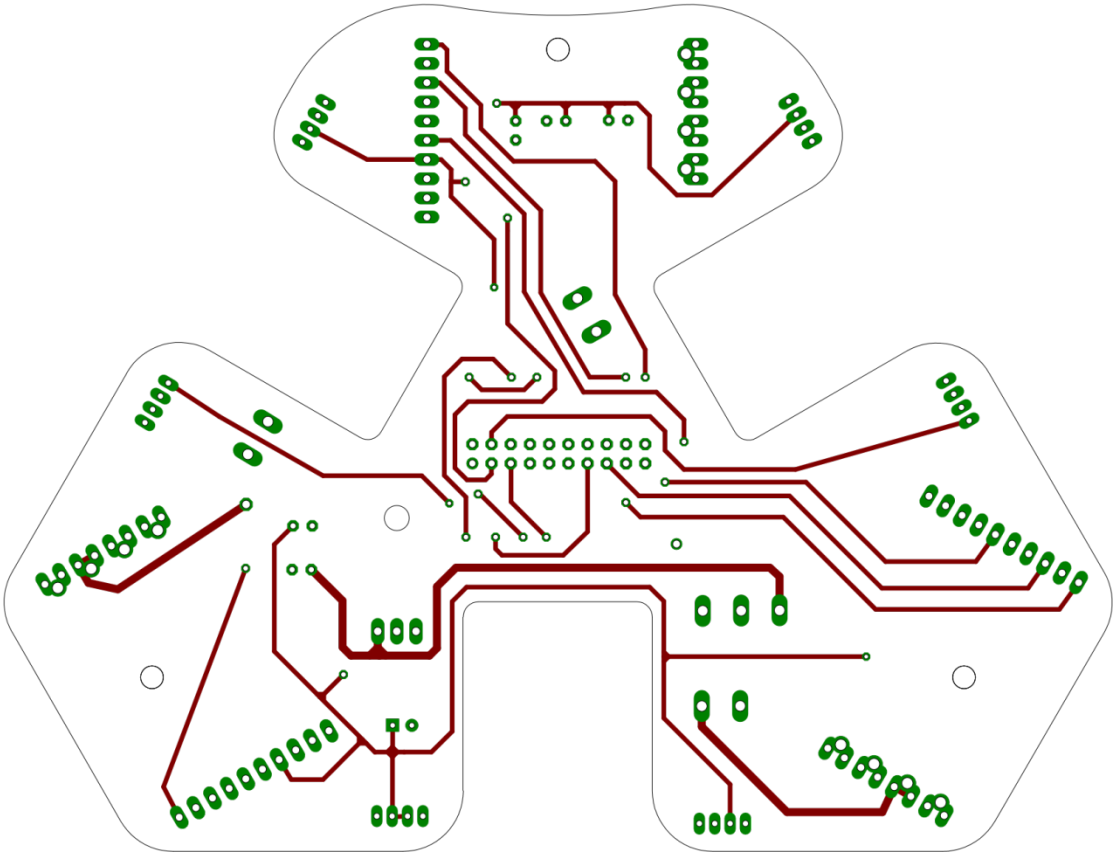


Motor Board PCB



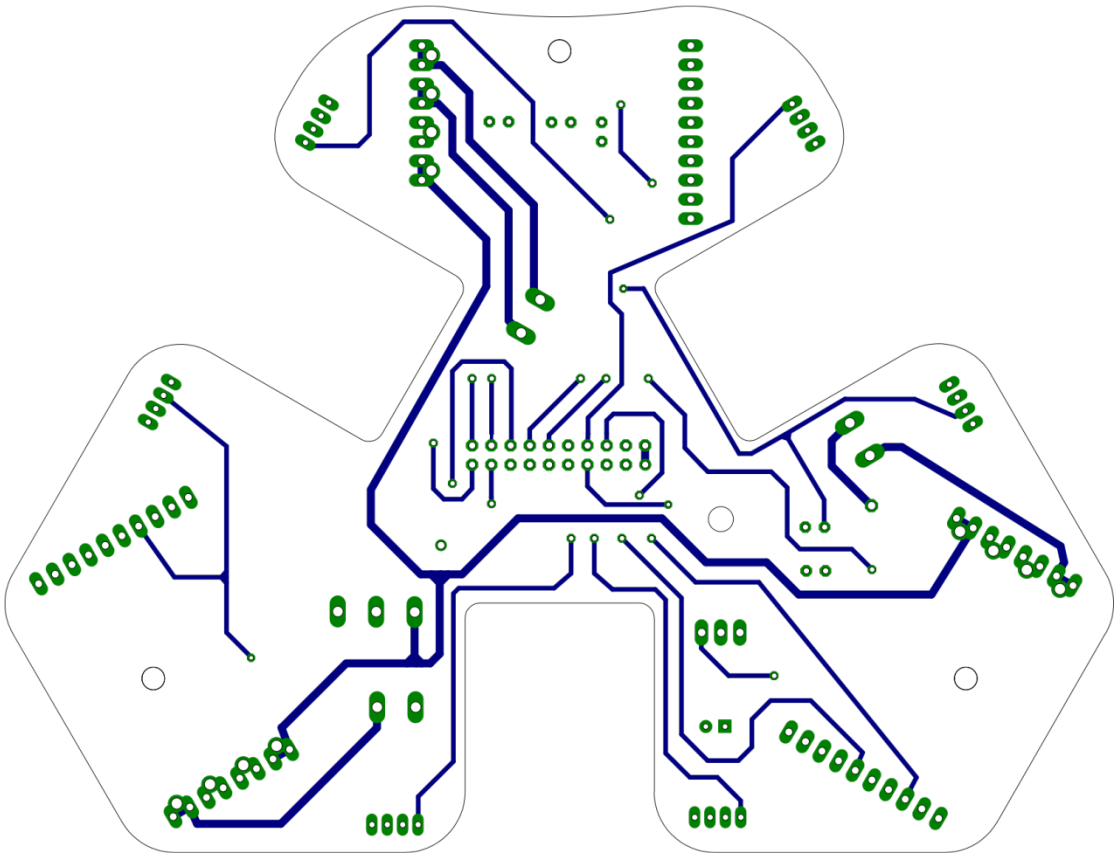
COMPONENTS LAYOUT

Motor Board PCB



TOP SIDE

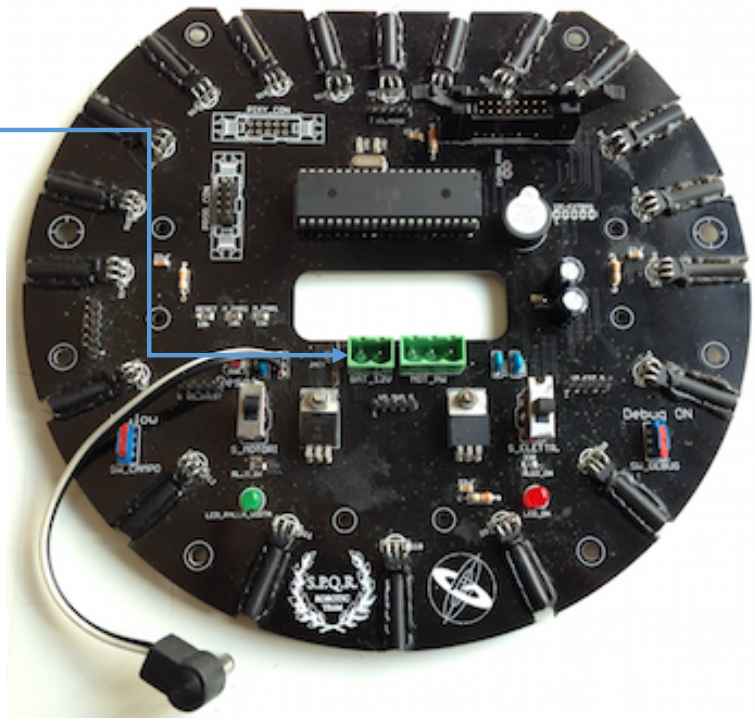
Motor Board PCB

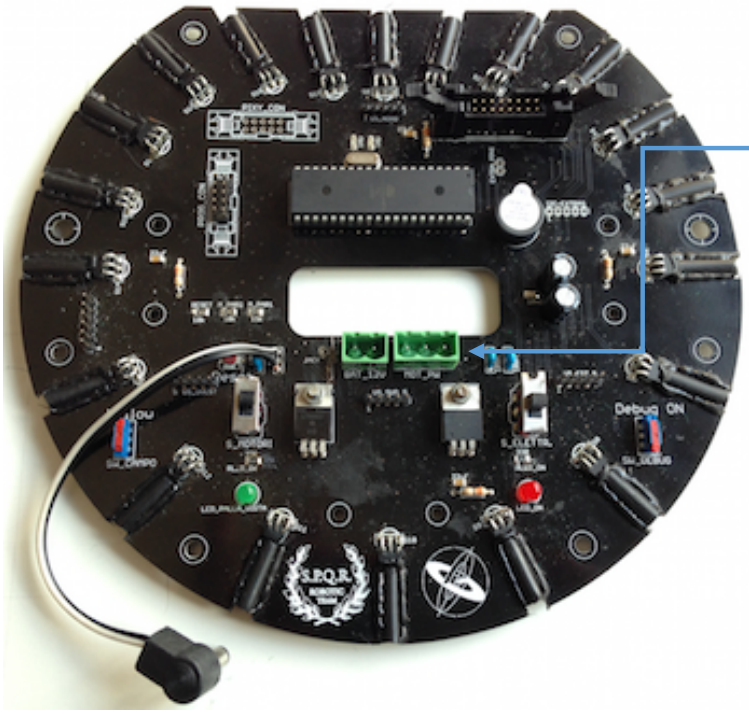


BOTTOM SIDE

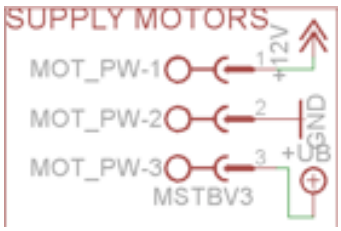
➢ *Ball Board*

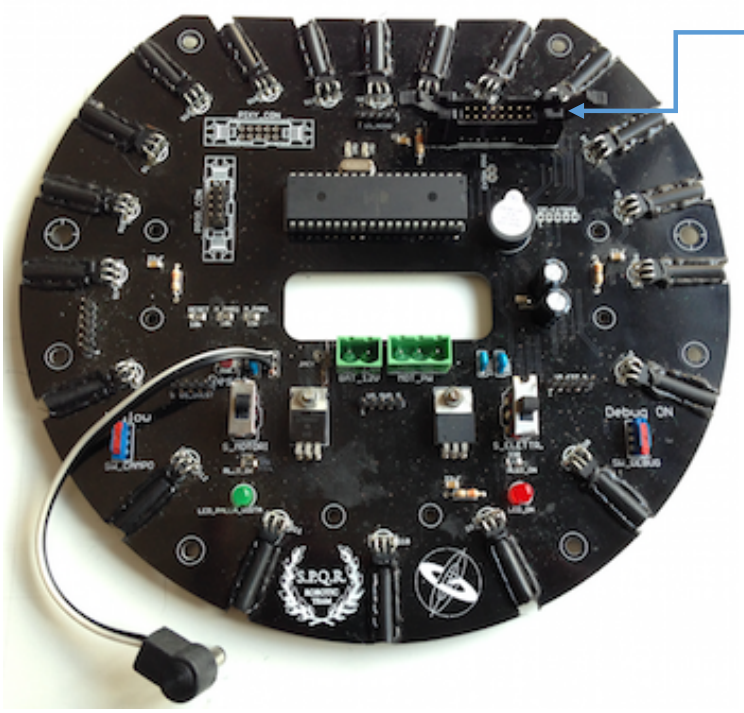
BAT_12V
2 pin Package MSTBV2
Directly connected to the battery



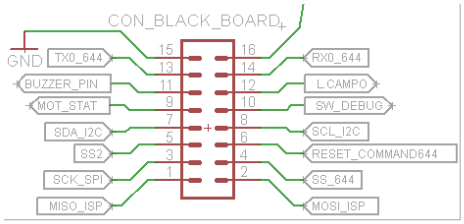


MOT_PW
3 pin Package MSTBV3
Tranfers 12V to the motor board
Strategically placed near



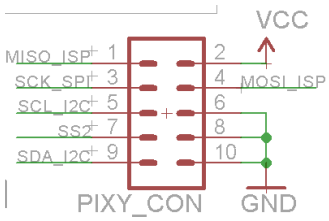


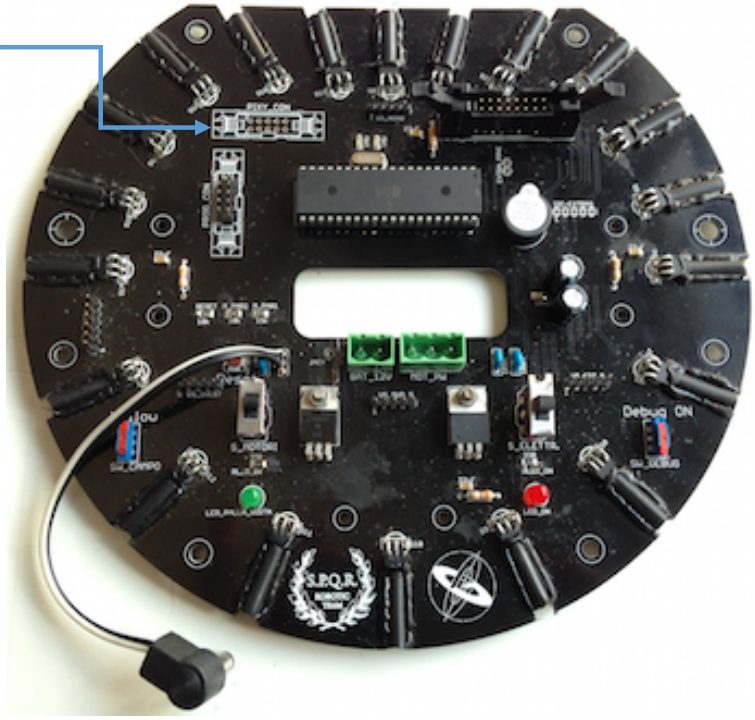
CON_BLACK_BOARD
Links the ball board to the Mega shield via SPI I2C & Serial Communications



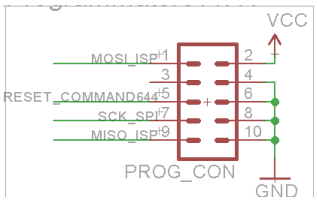
PIXY_CON

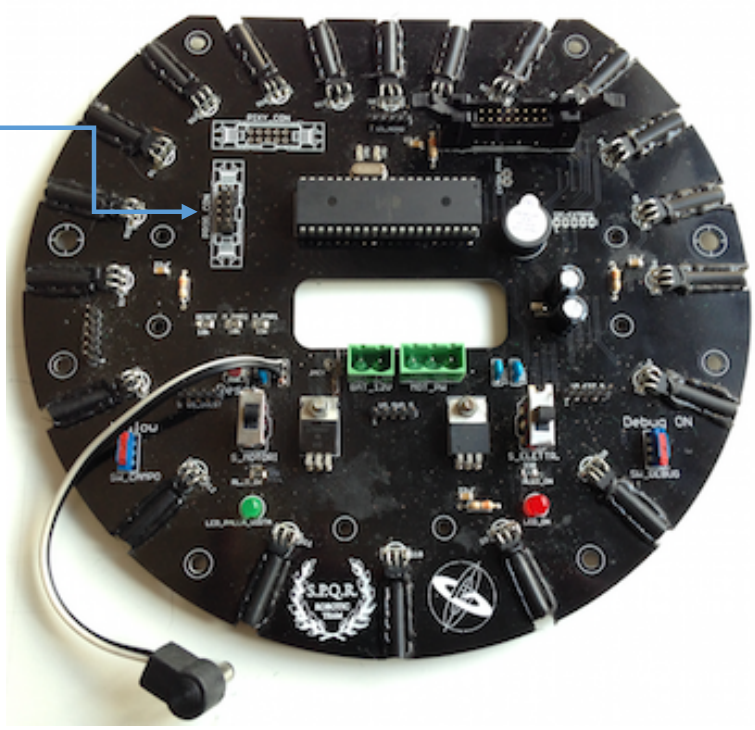
To connect PIXY camera



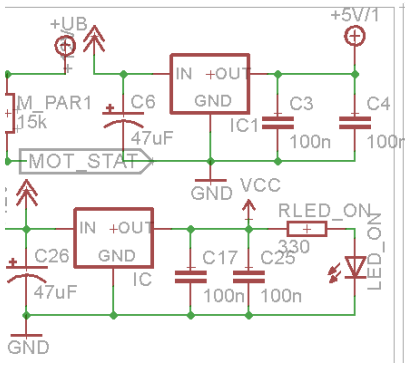


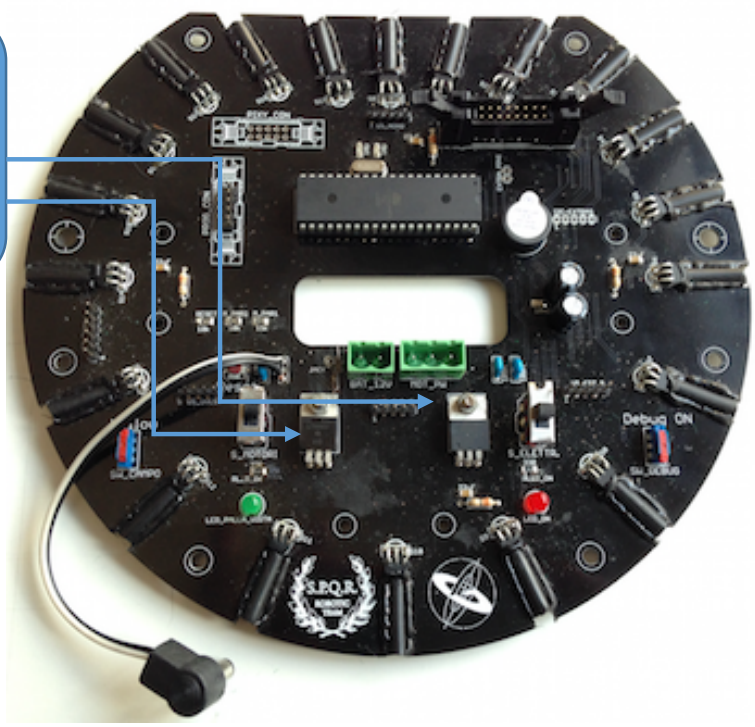
PROG_CON
ICSP connector to program the AT Mega 644p



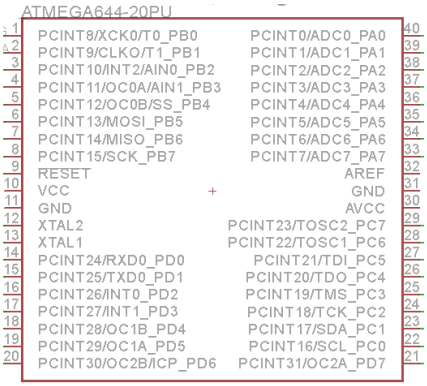


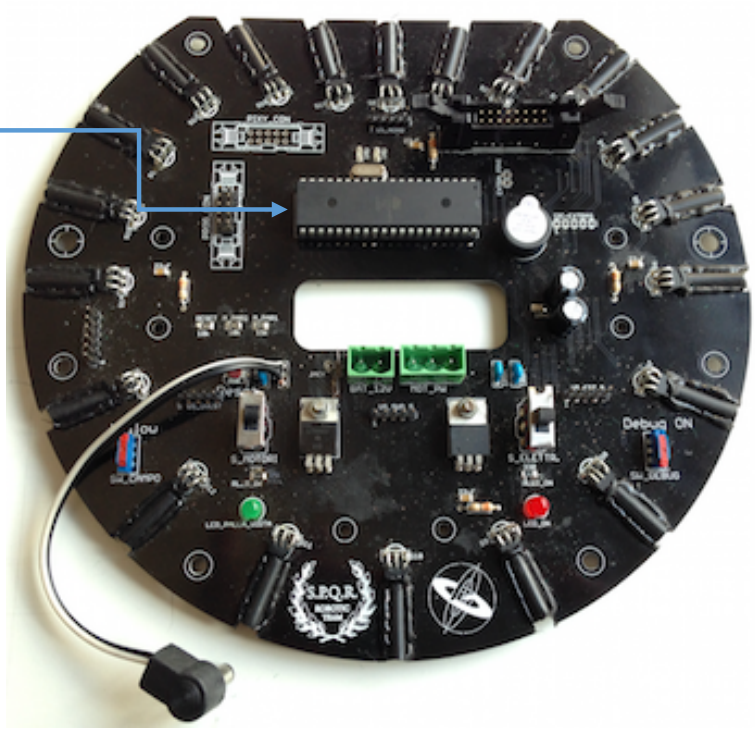
IC1 IC2 VOLTAGE REGULATORS
7805 5V to electronics
7808 8V to Arduino board



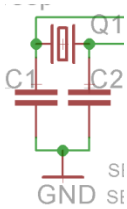


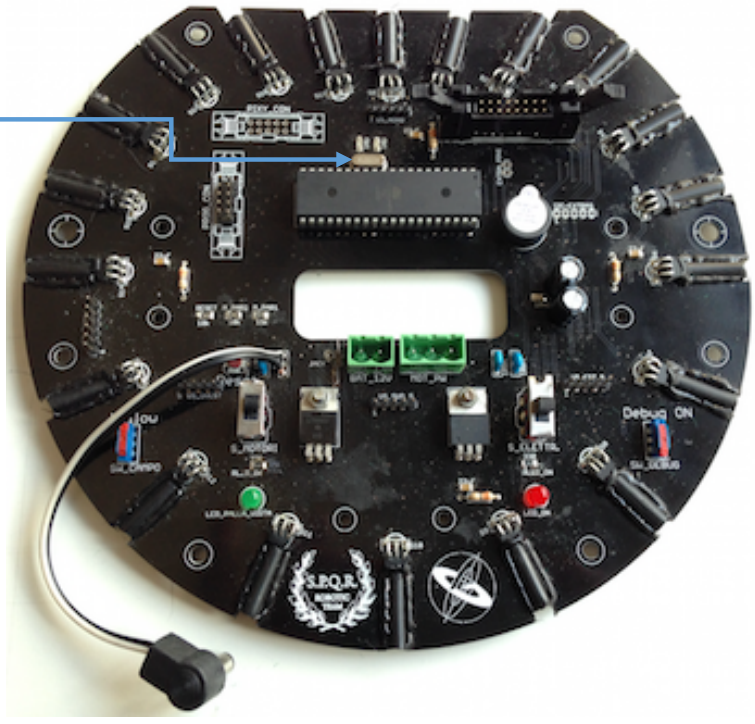
IC2 AtMega 644p
Package DIL40
Connected to ball sensors



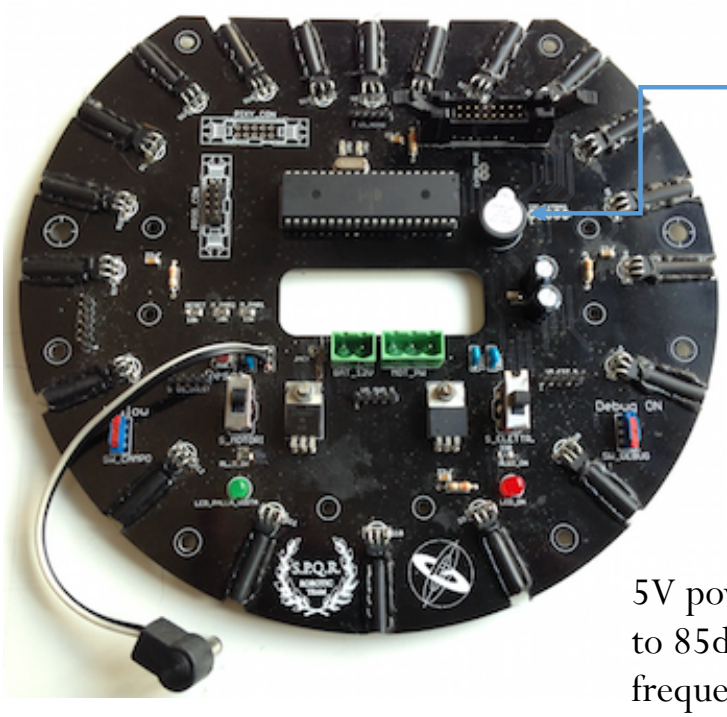


Q1: 20 MHZ
package HC49U-V
18 pF C1 C2
CLOCK OSCILLATOR

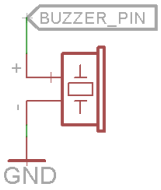




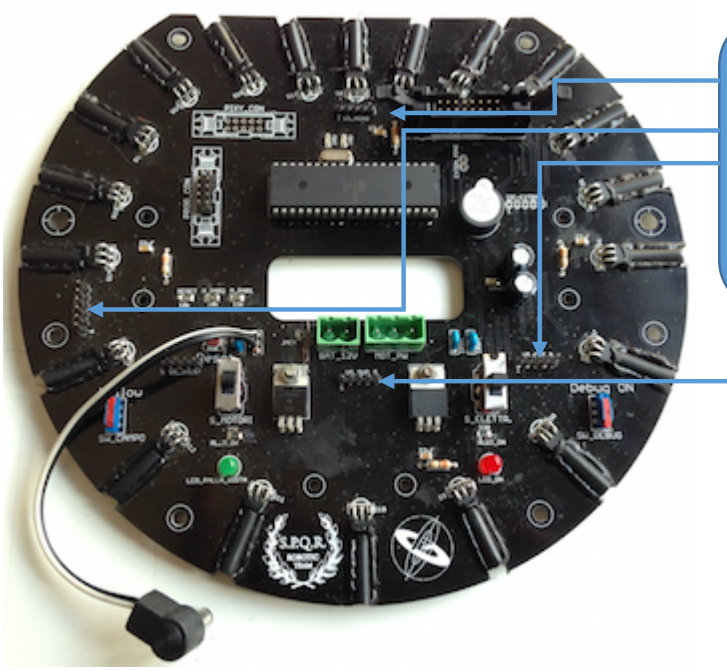
C1 e C2 package SMD `C1206`.



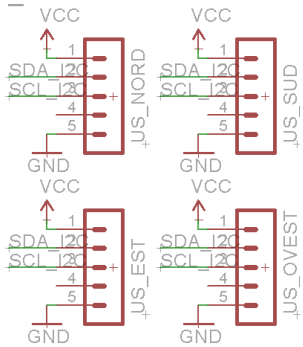
BUZZER
Beeps when the robot crosses a white line
Alerts at the end of compass calibration ends.



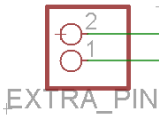
5V powered sends sound waves up to 85dB at 1.9kHz-2.9kHz frequency



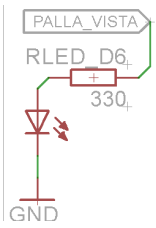
US CONNECTORS
5 pin strips
I2C signals
5V GND



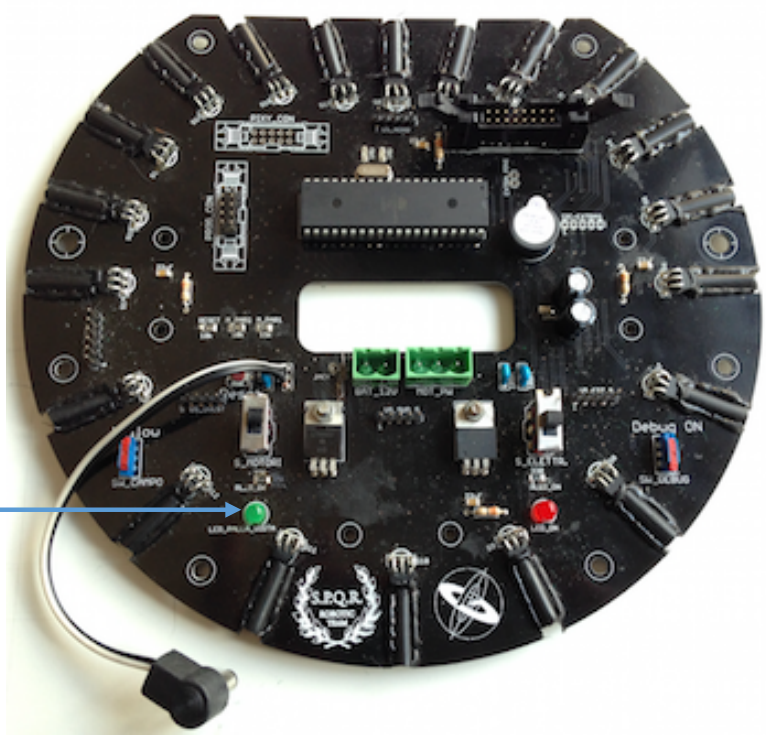
EXTRA_PIN1: 2 pins strip connected to ATMega 644p pin 3 & 4

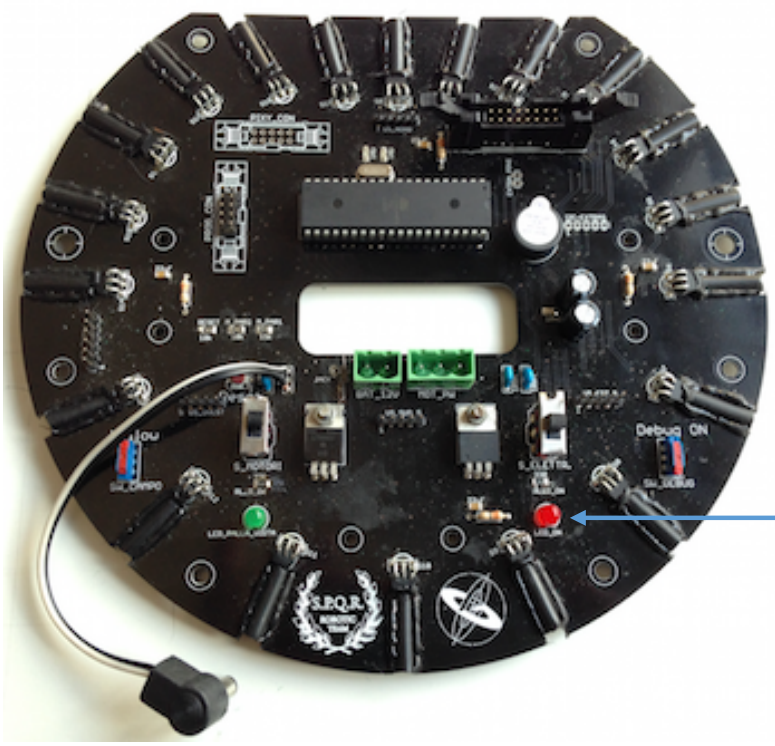


PALLA_VISTA
Lights when the ball is on sight

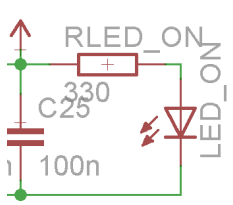


RLED 300 Ω SMD package M1206



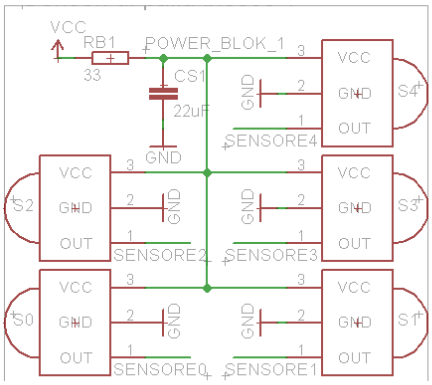


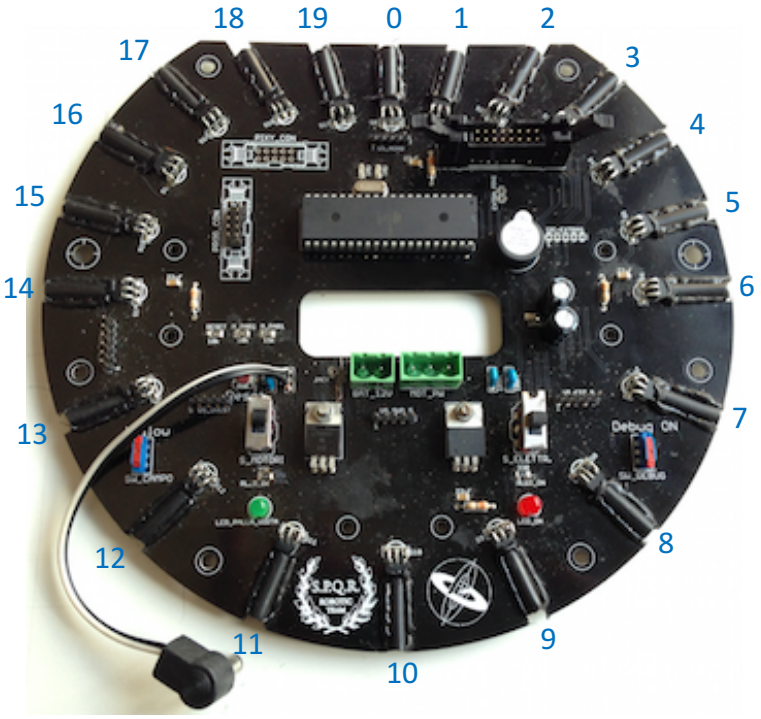
RLED_ON
Lights when the board is powered



Connected to ATMEGA644p pin 2

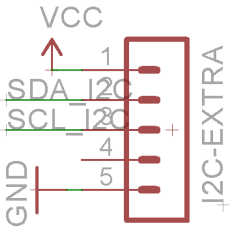
BALL SENSORS





`S0,S1,Sn.. S19=` 20 ball sensors package`PNA460XM`.
Strategically positioned on the board all sensors are inserted into the relative slots. To make our sensors more directional we have inserted at the end of each one a small black plastic pipe fixed to the board with hot glue.
As filter there are 33Ω resistors RB1, RB2, RB3, RB4 package `0207/10`, and 22uF capacitors CS1, CS2, CS3, CS4 package `SMD C1206`. We designed one filter for 5 ball sensors.

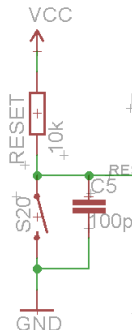
`I2C-EXTRA:`
5 pins strips to connect other I2C components

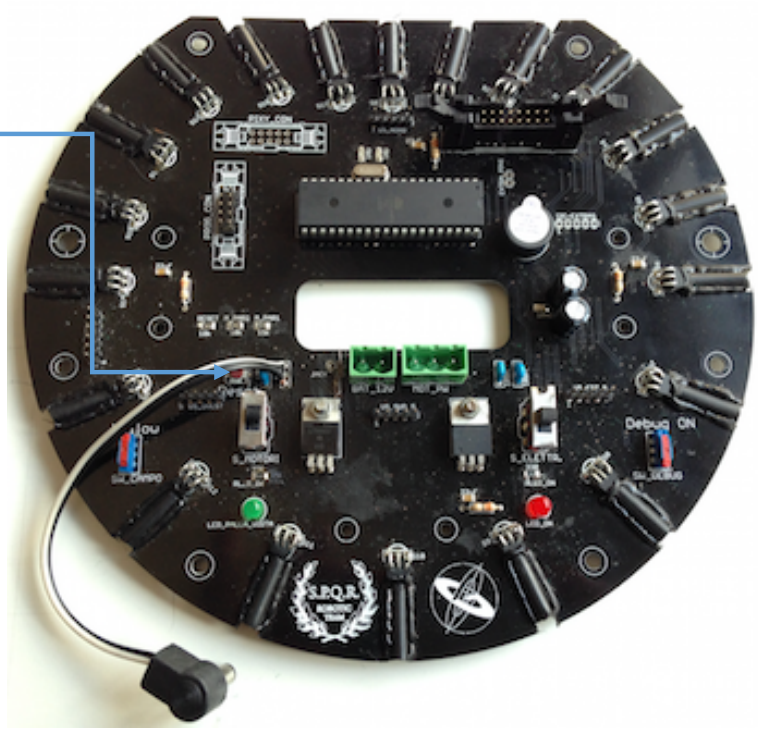


Switches On the Ball Board

S20 RESET

Momentary switch connected to the AT Mega 644p reset pin
It needs 3.1 N to act
In the centre of the board to avoid involuntary pressions

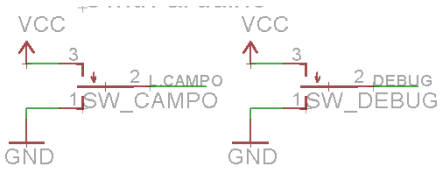


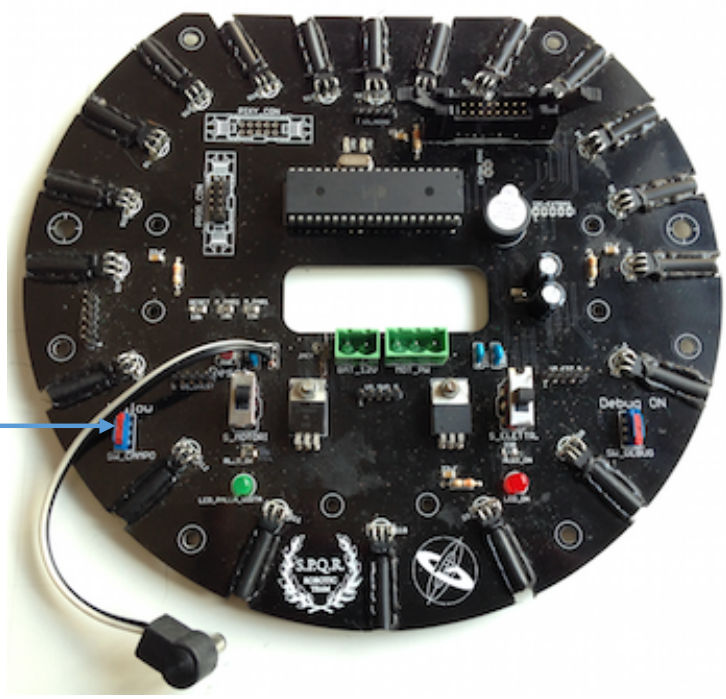


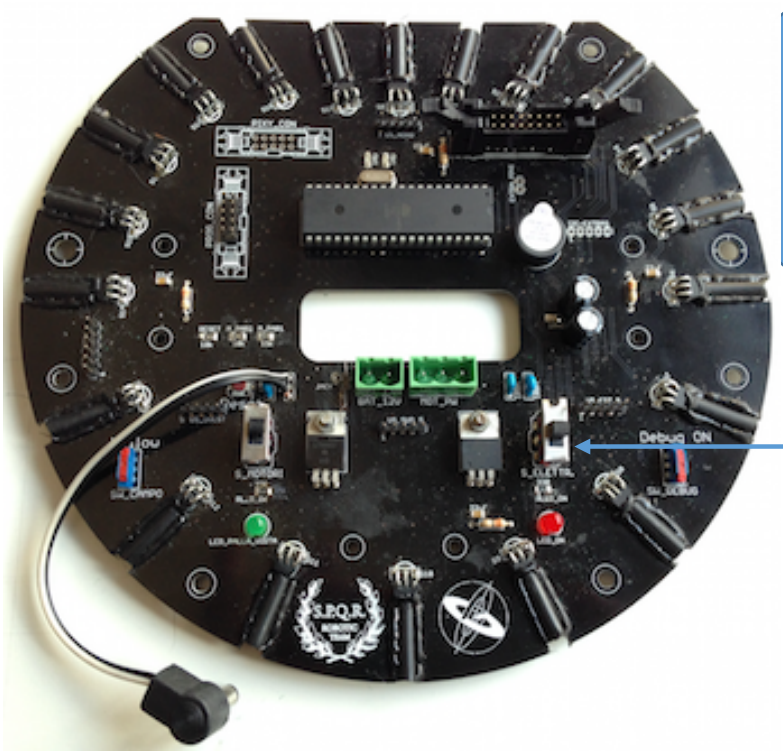
FIELD SWITCH

Package TL3XPO

Allows the robot yo know if the opponent goal is the yellow or the blue one
Easily reachable

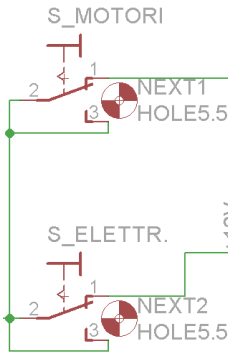






ELECTRONICS SW

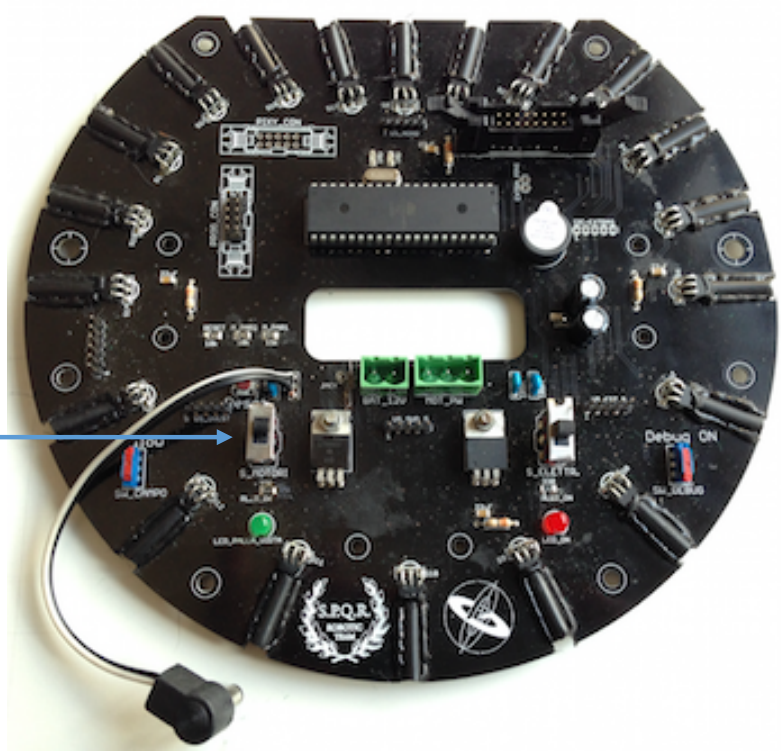
It is possible to power electronic components except motors & drivers

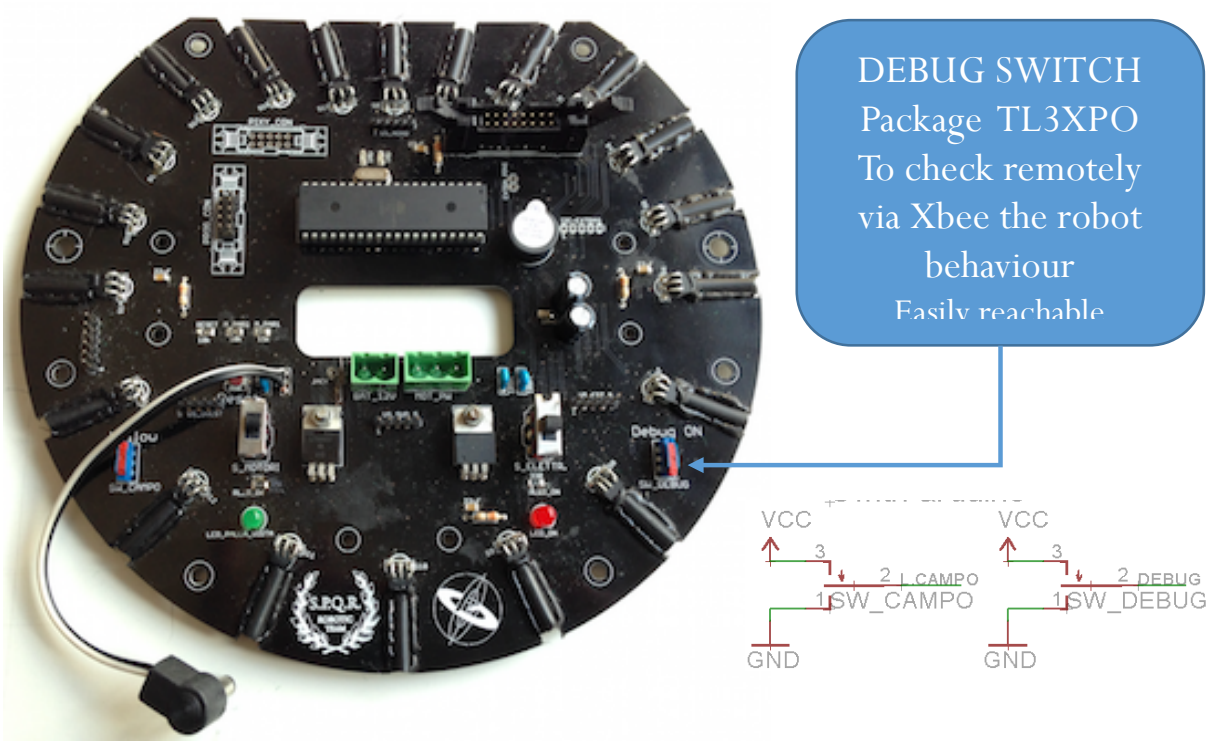


MOTOR SWITCH

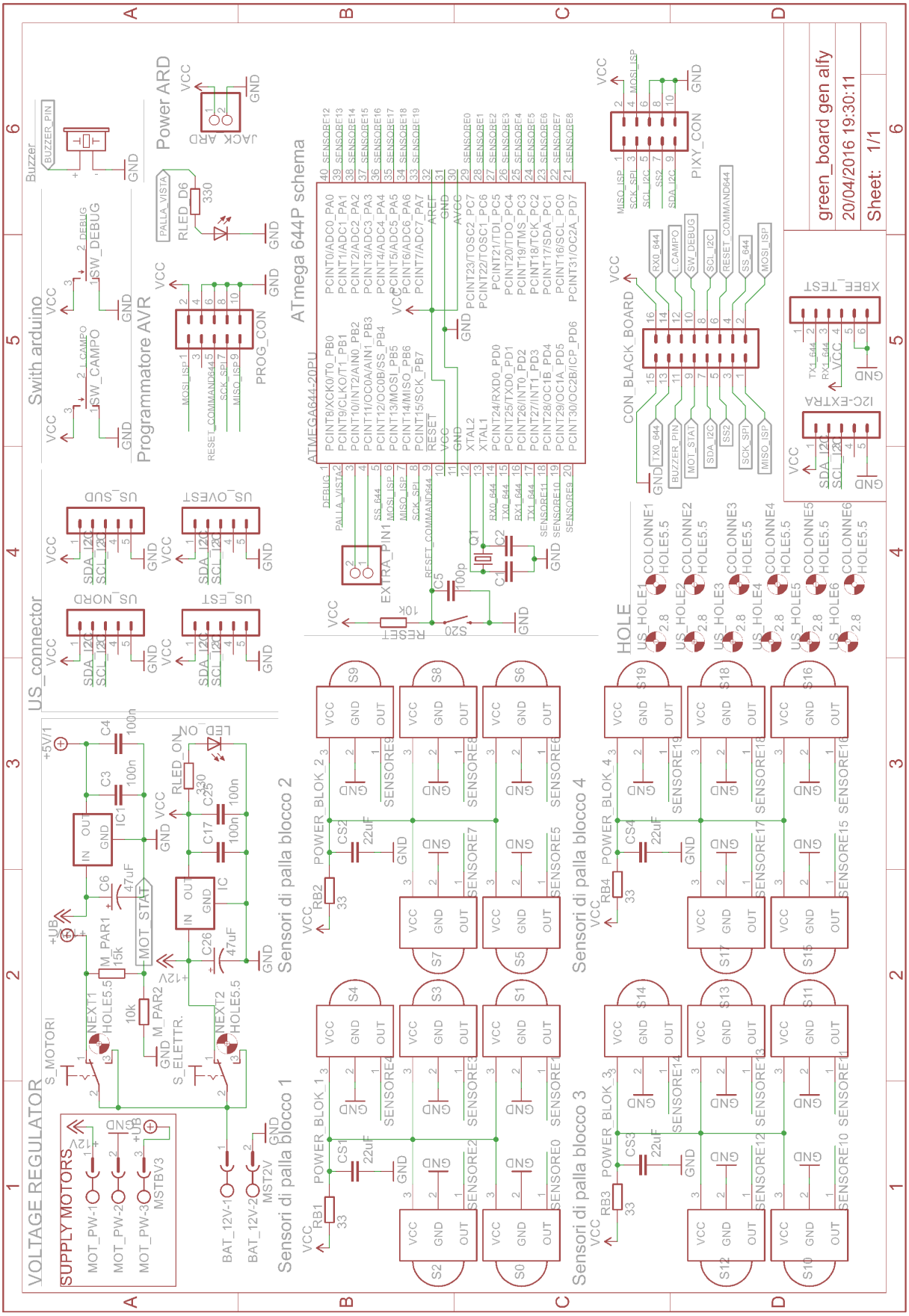
On = Motors powered

Off = Motors inactive

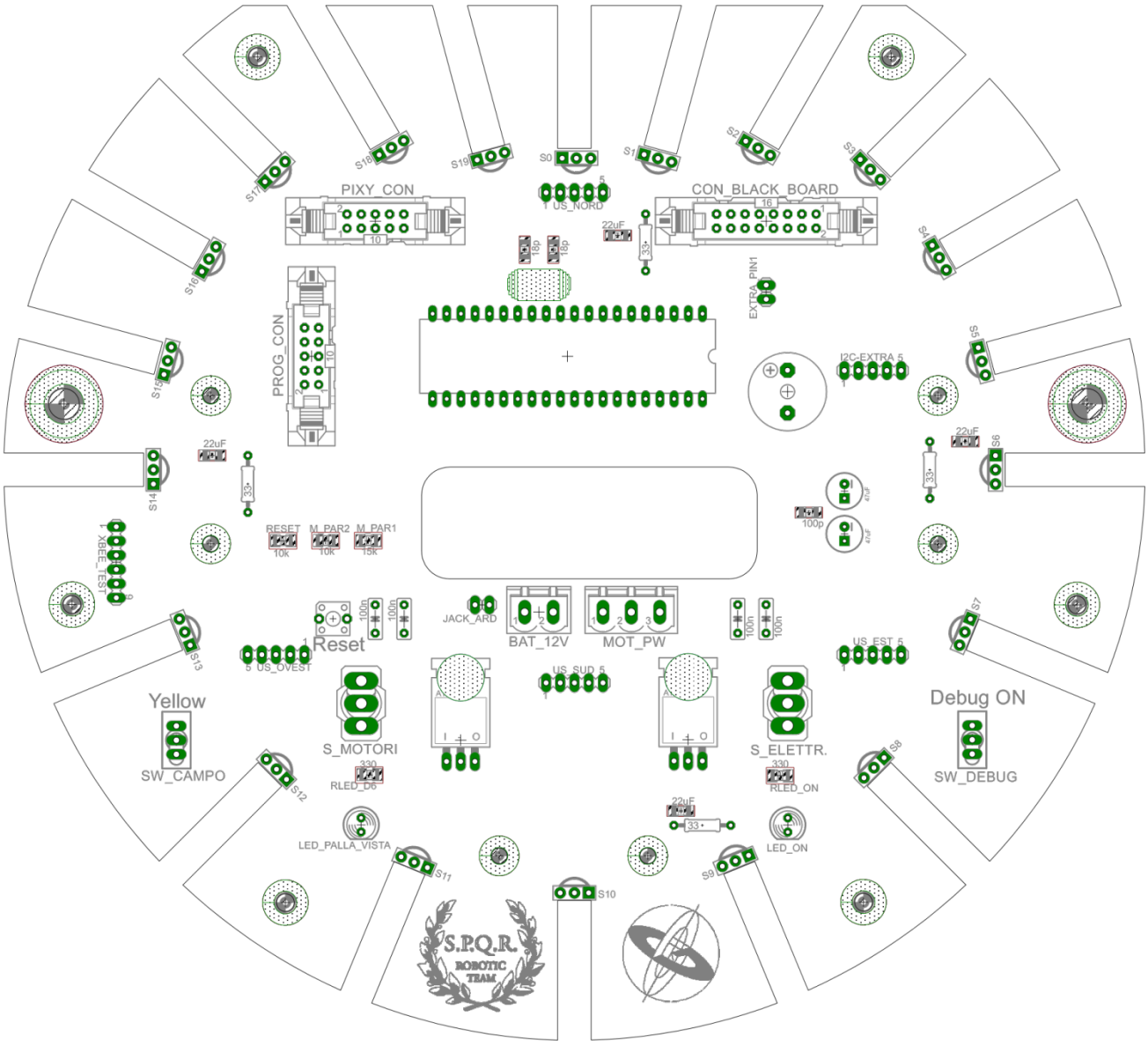



DEBUG SWITCH
Package TL3XPO
To check remotely
via Xbee the robot
behaviour
Easily reachable
Debug ON
VCC
3
2
1
SW_CAMPO
GND
VCC
3
2
1
SW_DEBUG
GND

Black Board Schematic

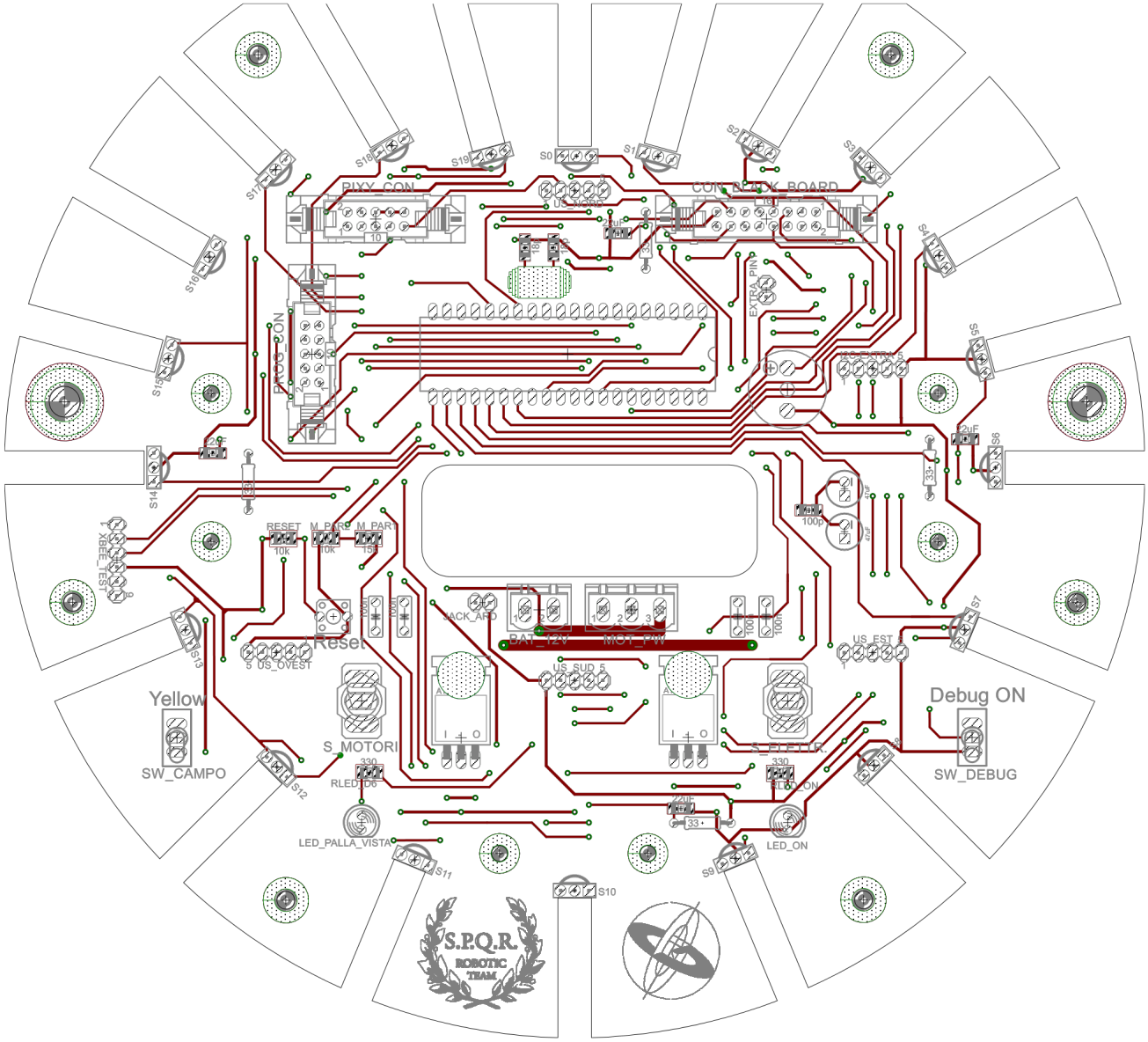


Ball Board PCB



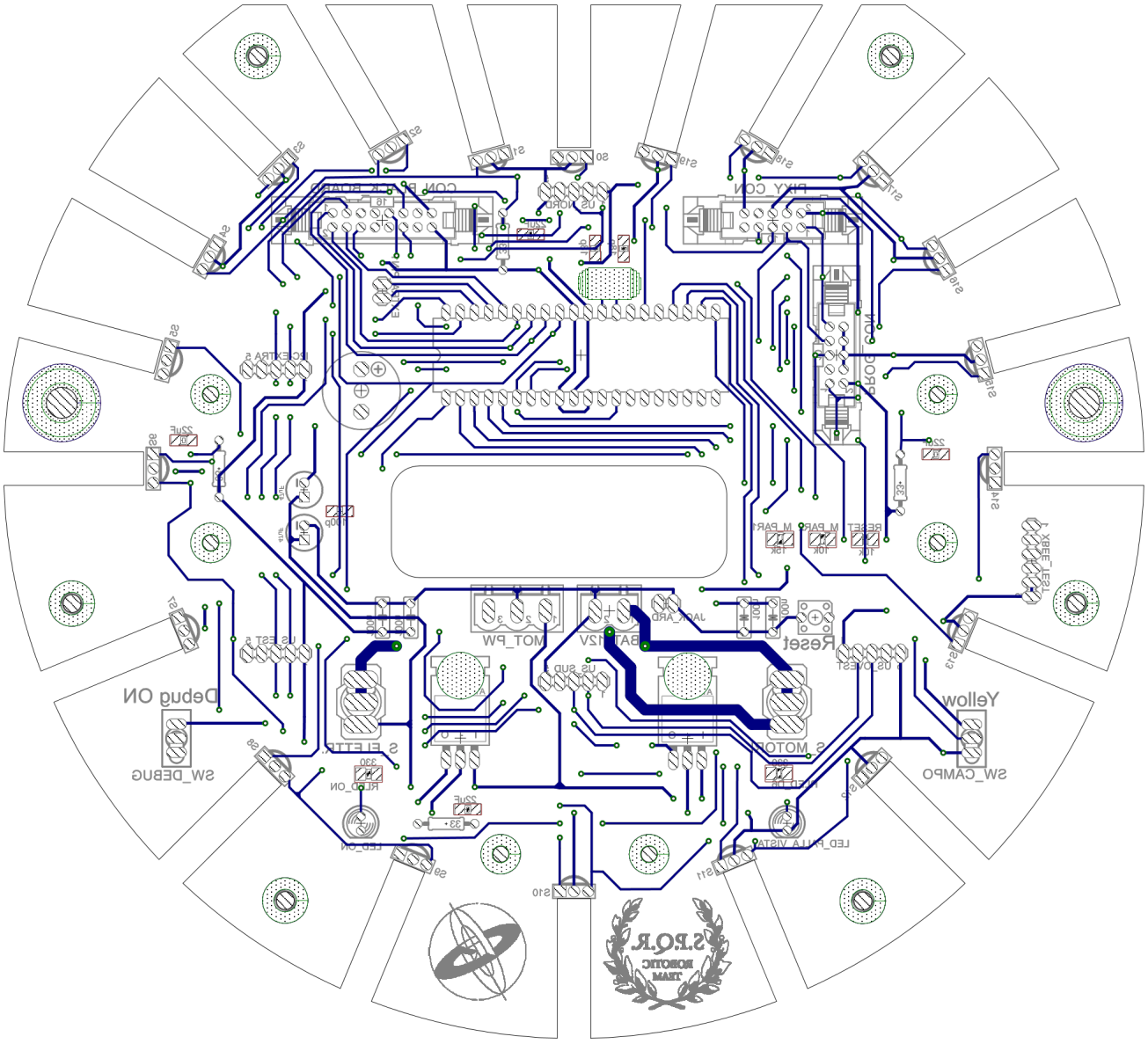
COMPONENTS LAYOUT

Ball Board PCB



TOP SIDE

Ball Board PCB

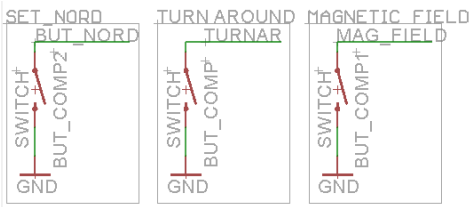


BOTTOM SIDE

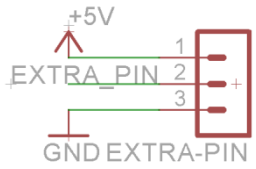
➢ *Mega shield*

ARDUINO_MEGA: The entire edge of the shield is filled with strips. Single line strips positioned along the longer side and double lines strips positioned on the short side. The pin layout is fully Arduino Mega compatible so you can fit the shield directly to the Arduino Mega.

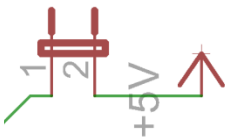
SET_NORD, MAGNETIC FIELD, TURN AROUND: These three buttons are used to set up the compass. These momentary switches, normally open, require 3.1N of strength to be operated. All of them are on the edge of the shield to facilitate their use.



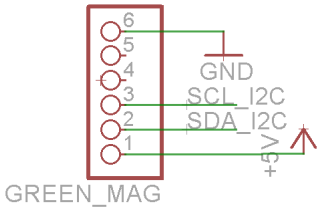
EXTRA-PIN: it is a three pins strip; at the moment it doesn't take any action. It is reserved for further utilization. Two pins are connected to GND and Vcc, respectively.



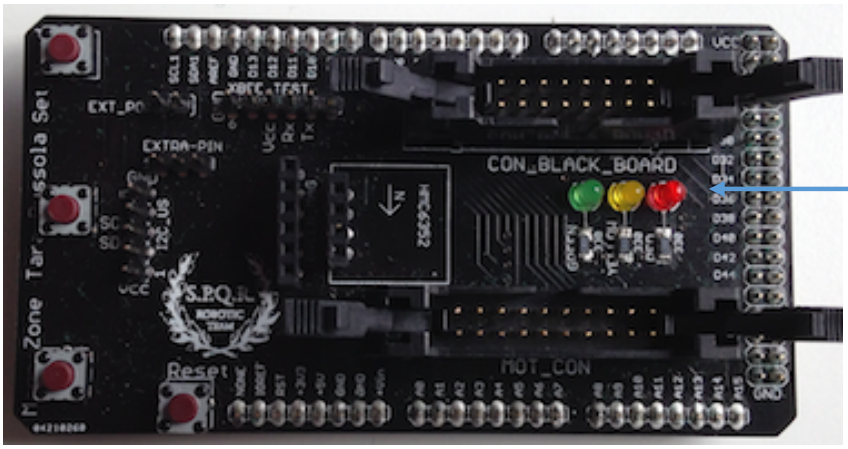
EXT_POW: it is a 2 pins jumper connected to the ball board connector. It permits to power up the Arduino Mega directly from the 5V level power produced by 7805 on the ball board. This is utilized when the internal voltage stabilizer of the Arduino Mega is overheated.



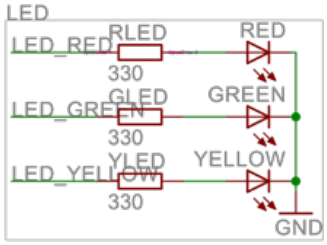
GREEN_MAG: a six pins connector on the board allows the usage of one of the possible compasses. The six pins carry the following signals: GND, nc, nc, SCL_I2C,SDA_I2C e 5V.

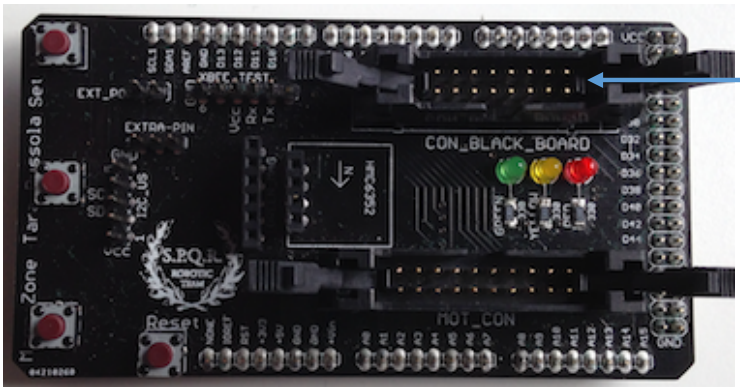


-

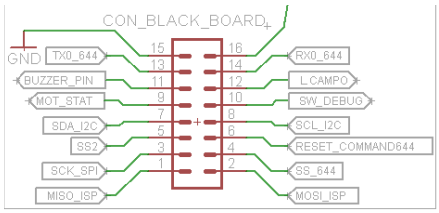


LEDS
3 mm Leds
Show Software Status

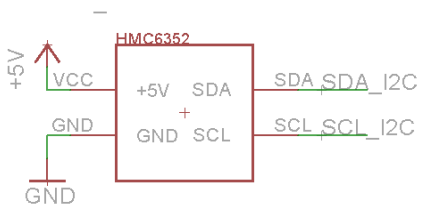


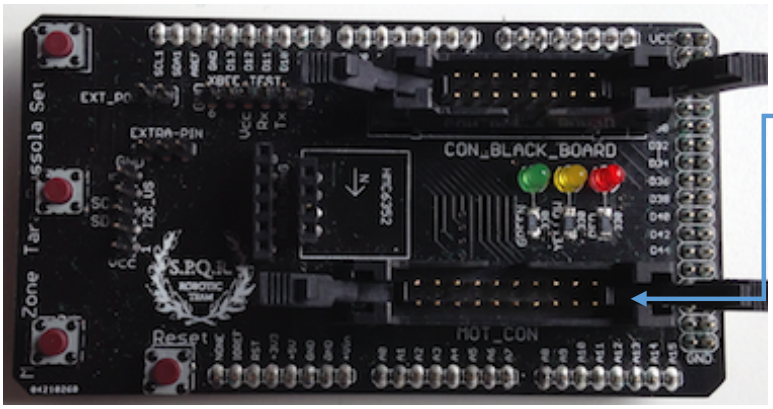


CON BALL BOARD
16 pins 8 +8
Serial SPI I2C
communications

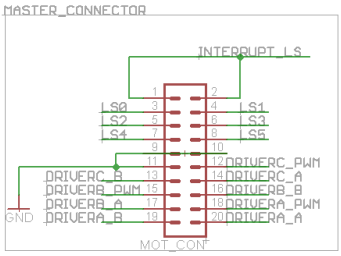


HMC6352: another four pins connector for a compass, carrying the following signals, is present on the board: GND, SCL_I2C, SDA_I2C e 5V. The specific position of these connectors has been designed to obtain the better positioning of the compasses in the layout of the robot.

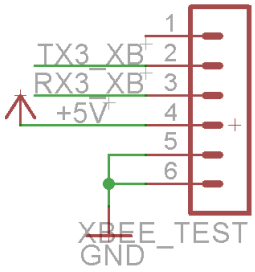


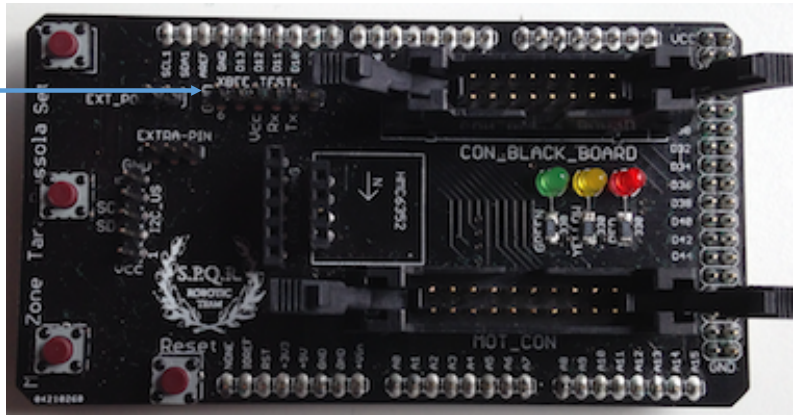


MOTOR CONNECTOR
20 pins 10 +10
Send Signals to motors



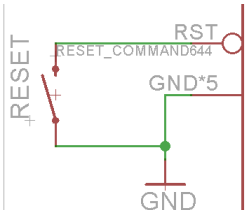
XBEE_TEST
To connect the DEBUG STATION via
Pin TX3 RX3 AVR2560

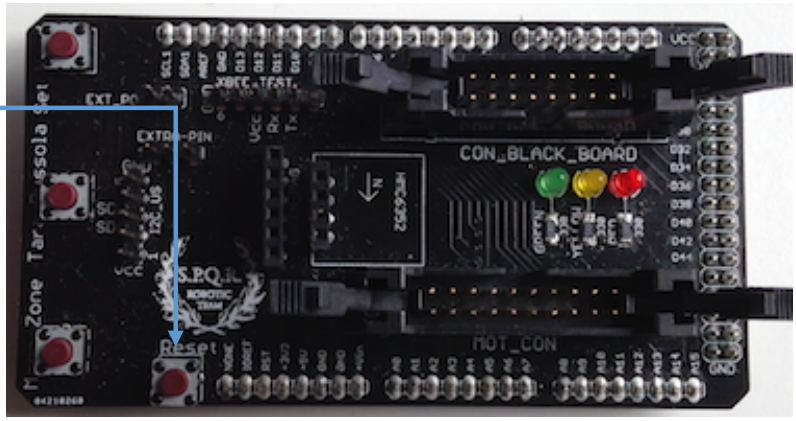




-

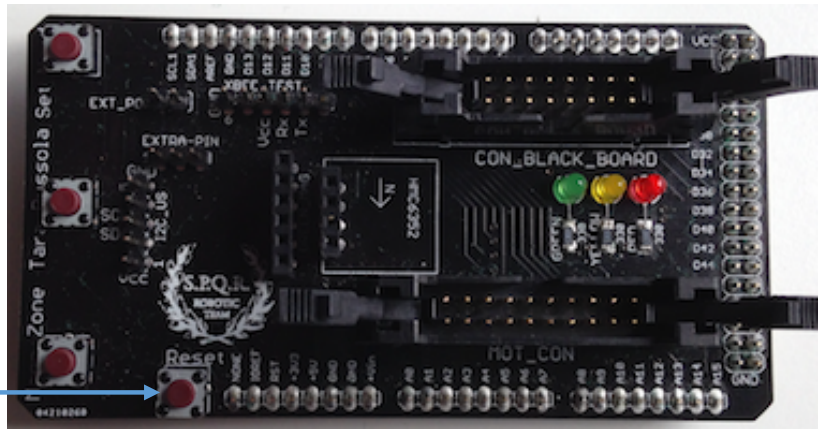
RESET
Monentary Switch
Resets Arduino Mega



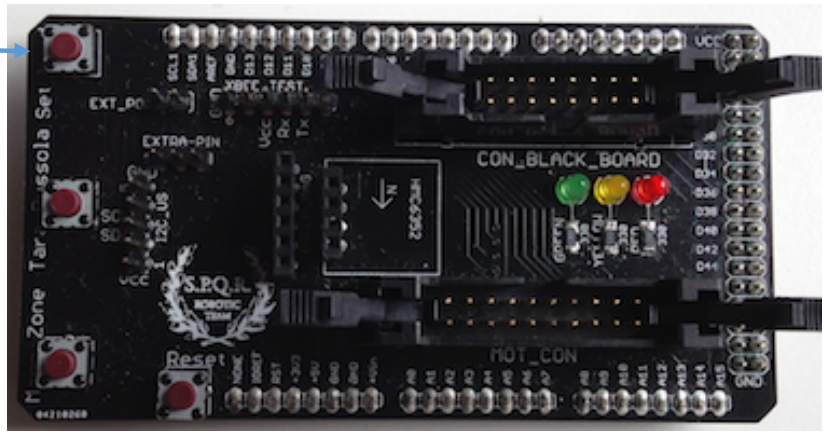


➢ *Switches On the Mega Shield*

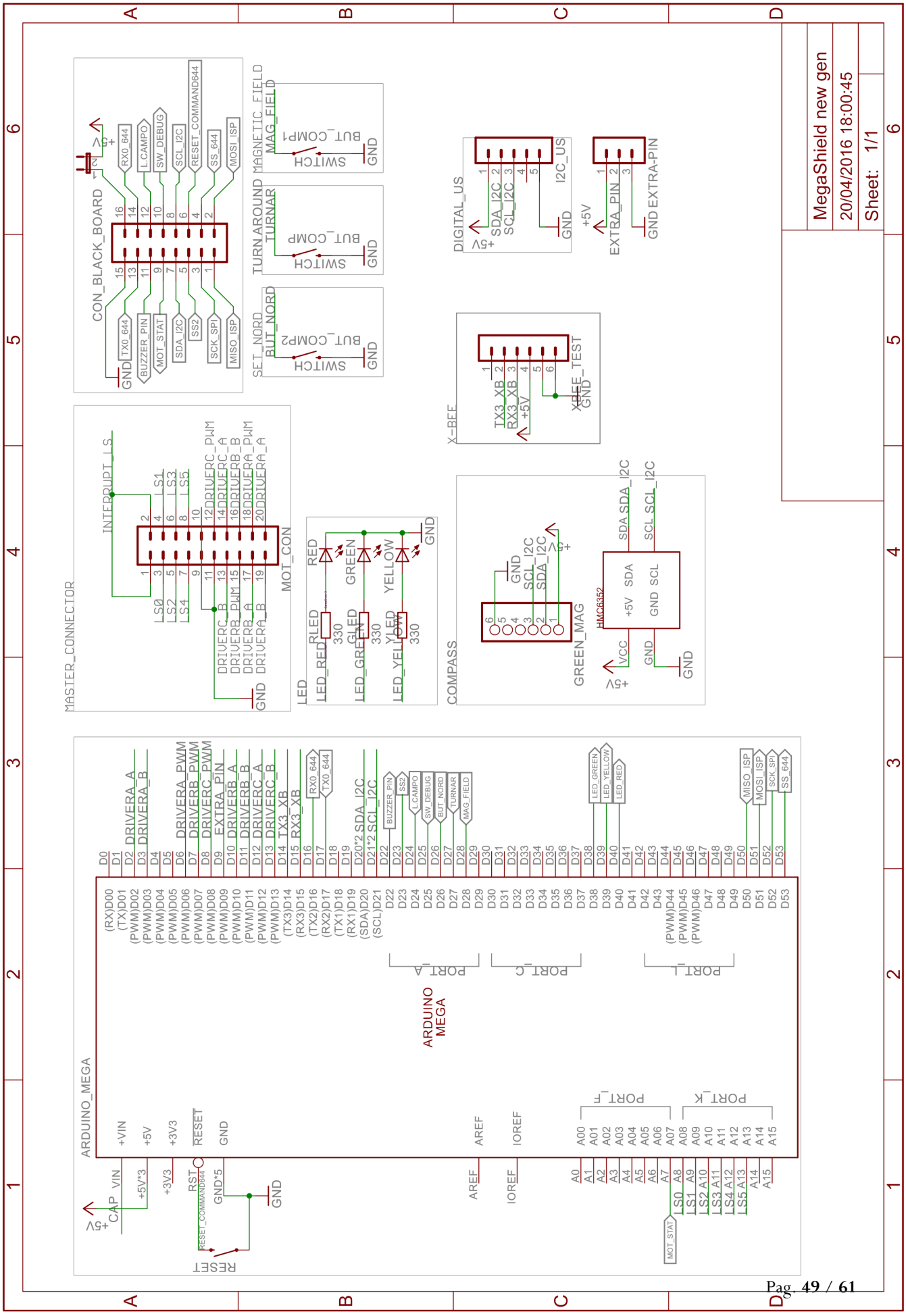
Reset
Connected to the pin reset of the AVR 2560 on the Arduino Mega



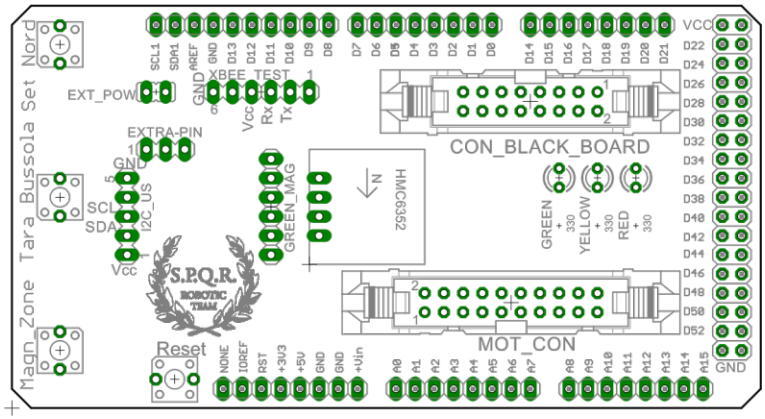
NORD
Set the opponent goal as «Virtual North»



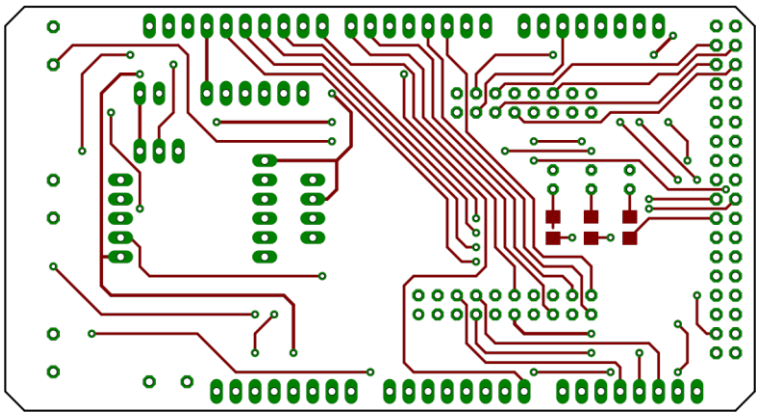
Schema elettrico Mega Shield



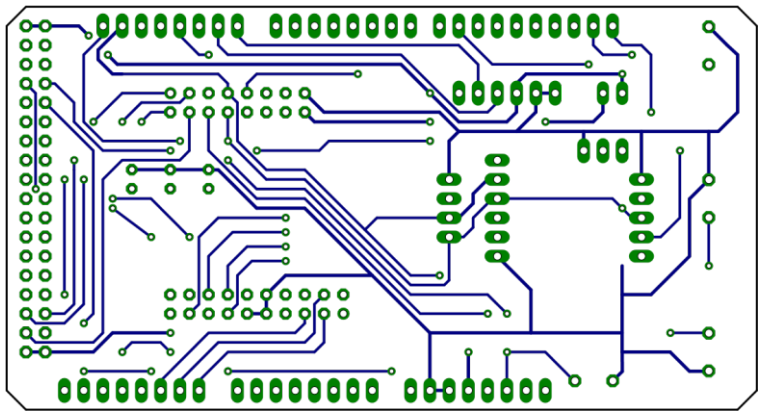
PCB Mega Shield



COMPONENTS LAYOUT



TOP SIDE



BOTTOM SIDE

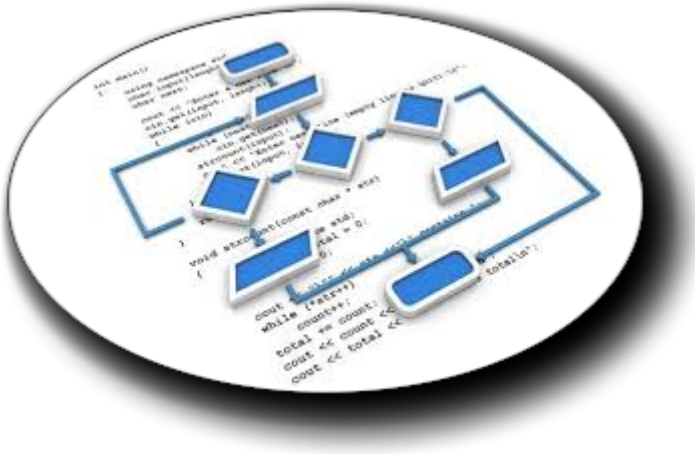
Software design

- The main board program is in Arduino C
- The ball sensor board is programmed in Arduino C
- The Debug Station program is in Processing

Our robots software is quite different

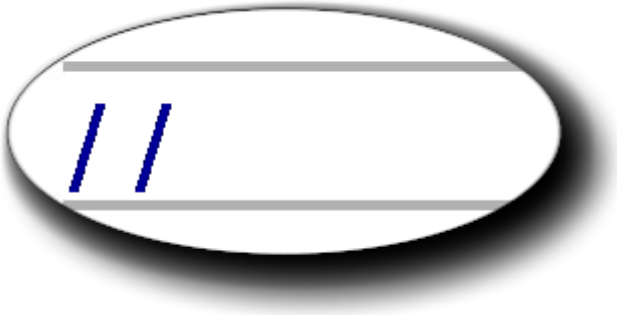
- The Forward robot tries to attack and score if it "sees" the ball
- The goal keeper moves forward only when the ball is near

Different software prevents robots from hinder each other

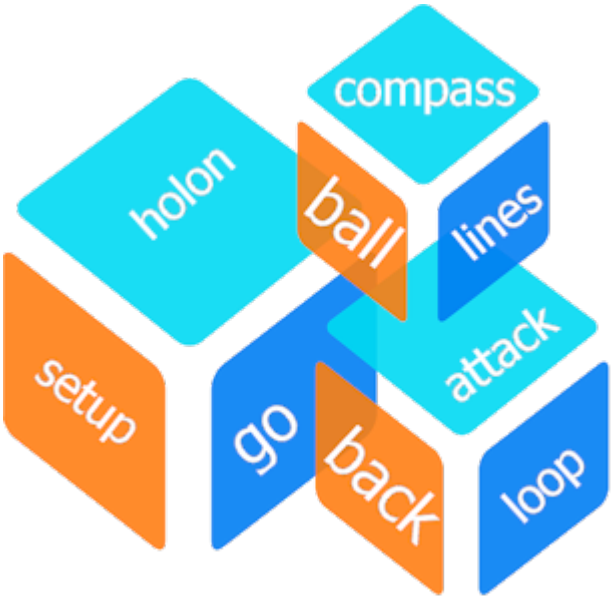


Each part of our software has been accurately documented.

Flow chart has been drawn for each routine.



Our code has plenty of comments because it is continuously evolving and keeping track of changes is a must.

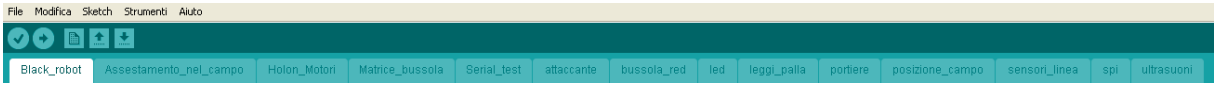


Modularity is the key for a good software
For each task we built a separate function

In this way it is possible to change part of the hardware and its relative driver without affecting the entire program

Our debug station in Processing language is realized in OOP.

Software Macro Areas.



There are 14 relevant tasks that the robot performs: for each task we have planned a different Tab in the Arduino IDE

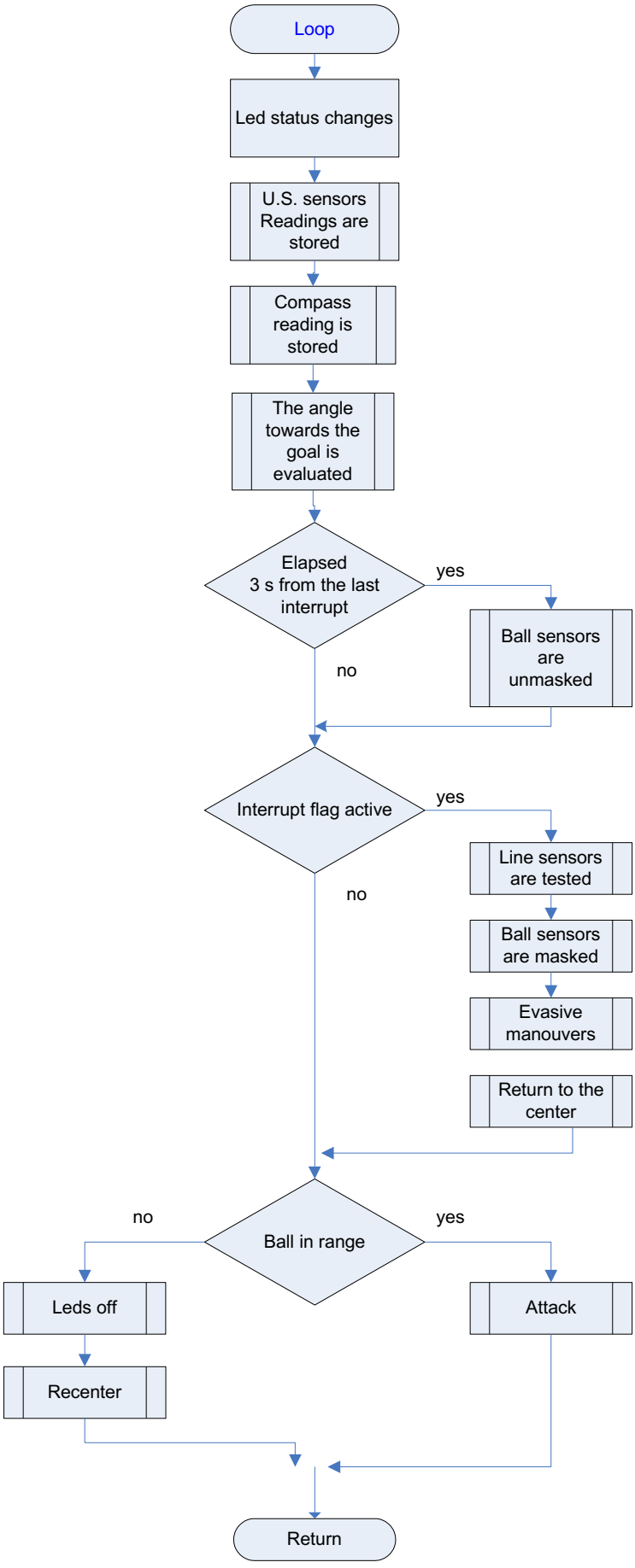
Black_robot	***Initialization Setup & Loop***
Assestamento_nel_campo	***Checks Robot Orientation***
Holon_Motori	***Robot holonomic movements***
Matrice_bussola	***Magnetic inteferences compensation***
Serial_test	***Local tests***
Attaccante	***Forward Strategy***
Bussola_red	***Compass management***
Led	***Alerts management***
Leggi_palla	***Acquires ball position***
Portiere	***Goal keeper strategy***
Posizione_campo	***Statistic robot position***
Sensori_linea	***Line sensors analysis & Interrupt***
Spi	***SPI communication***
Ultrasuoni	***Sonar Range Finder management***

In the following pages there is a detailed documentation of

- **Variables**
- **Constants**
- **Functions**

for each tab and the loop flow chart .

Main Loop Flow Chart



TAB "Assestamento nel campo" (Checks Robot Position)

Functions				
NAME	**PARAMETERS**	**CALLED FUNCTIONS**	**PURPOSE**	**OUTPUT**
int ricentra_serial( )	*fun_xbee,gradi_norm, VEL_RIC[MIN],VEL_RIC[MIN],MIN ,MAX*	*fermo( ), ruota ( )*	*Makes the robot be facing the opponent goal and in the Middle of the field (Called by Debug)*	*No use*
void riallinea_in_gioco( )	*gradi_norm,VEL_RIC[MIN], VEL_RIC[MIN], MIN,MAX*	*ruota( )*	*Makes the robot be facing the opponent goal*	-
void ritorna_CC( )	*ZONA_*,FASCIA_*,gradi_norm ,VEL_RIC[MED], sonar_dx, sonar_sx*	*go( ), riallinea_in_gioco( ), interpola_pos( ), fermo( )*	*Makes the robot go to the Middle of the field*	-
void ritorna_porta( )	*sonar_px,sonar_dx,sonar_sx, VEL_RIT[MED],VEL_RIT[MAX], gradi_norm,ZONA_*,FASCIA_**	*go( ),riallinea_in_gioco( ), interpola_pos( ), ricentra_serial( )*	*Makes the robot go to ZONA_SUD (8)*	-
boolean ritorna_fascia_sud( )	*ZONA_*,FASCIA_*,VEL_RIT[MAX] ,VEL_RIT[MED],gradi_norm, sonar_dx,sonar_sx*	*go( ),riallinea_in_gioco( ), interpola_pos( ),*	*Makes the robot go to FASCIA_SUD (15).*	*No use*

TAB "Assestamento nel campo" (Checks Robot Position)

Variables		
NAME	**PURPOSE**	**USED IN**
*FASCIA_**	*Robot position in the field FASCIA_ NORD,SUD,EST,OVEST,MIDDLE_X,MIDDLE_Y*	*void ritorna_CC( )* *void ritorna_porta( )* *boolean ritorna_fascia_sud( )*
fun_xbee	*No more used*	*int ricentra_serial( )*
gradi_norm	*Degree of the opponent goal versus the real North*	*int ricentra_serial( )* *void riallinea_in_gioco( )* *void ritorna_CC( )* *void ritorna_porta( )* *boolean ritorna_fascia_sud( )*
sonar_dx	*Reading from the right sonar.*	*void ritorna_CC( )* *void ritorna_porta( )* *boolean ritorna_fascia_sud( )*
sonar_px	*Reading from the rear sonar*	*void ritorna_porta( )*
sonar_sx	*Reading from the left sonar.*	*void ritorna_CC( )* *void ritorna_porta( )* *boolean ritorna_fascia_sud( )*
VEL_RIC[MED/MIN/MAX]	*Robot speed when the robot tries to go in the middle of the field*	*int ricentra_serial( )* *void riallinea_in_gioco( )* *void ritorna_CC( )*
VEL_RIT[MED/MIN/MAX]	*Robot speed when the robot tries to go in the middle of the field after an interrupt*	*void ritorna_porta( )* *boolean ritorna_fascia_sud( )*
*ZONA_**	*Robot position in the field NORD,NORD_EST,NORD_OVEST,CENTRO,EST,OVEST,SUD, SUD_OVEST,SUD_EST*	*void ritorna_CC( )* *void ritorna_porta( )* *boolean ritorna_fascia_sud*

TAB "HOLON MOTORI" (Robot holonomic movements)

Functions				
NAME	**PARAMETERS**	**CALLED FUNCTIONS**	**PURPOSE**	**OUTPUT**
void go(angolo, velocita, my_bussola)	*signed int angolo, signed int velocita, int my_bussola*	*mot( )*	*Computes the correct speeds for each motors so the robot goes at the desired angle knowing the robot orientation*	-
void init_holon( )	*angolo1,angolo2,angolo3, ANGOLOMOT1,ANGOLOMOT2, ANGOLOMOT3, matrice[0/1/2][0/1/2]*	*sin( ), cos( )*	*Computes X/Y components of each motor*	-
void init_motori( )	*INA_MOT[1/2/3], INB_MOT[1/2/3], PWM_MOT[1/2/3]*	*pinMode( )*	*Set in OUTPUT the pins connected to motor drivers*	-
void mot(mot,vel)	*byte mot, int vel*	*digitalWrite( ), analogWrite( )*	*Tuns on the selected motor at the desired spedd*	-
float radianti(gradi)	*float gradi*	-	*Degrees to radiants converter*	*Radiants*
float gradi(radianti)	*float radianti*	-	*Radiants to degrees converter*	*Degrees*
void fermo( )	-	*mot( )*	*Full halt*	-
void ruota (vel)	*int vel*	*mot( )*	*Makes the robot to turn at desired speed*	-

TAB "HOLON MOTORI" (Robot holonomic movements)

Variables		
NAME	**PURPOSE**	**USED IN**
matrix[3] [3]	*Being the field ideally devided in 9 zones, this array contains the probability that the robot has to be in that zone*	*void go(angolo, velocita, my_bussola)* *void init_holon( )* *Init_holon()*
angolo1	Variabile necessaria per risolvere il problema matematico del calcolo della direzione che viene impostata nella matrice[3][3].	*init_holon( )*
angolo2	Variabile necessaria per risolvere il problema matematico del calcolo della direzione che viene impostata nella matrice[3][3].	*init_holon( )*
angolo3	Variabile necessaria per risolvere il problema matematico del calcolo della direzione che viene impostata nella matrice[3][3].	*init_holon( )*
vx	*Projection on the x axis of the robot speed*	*go( )*
vy	*Projection on the y axis of the robot speed*	*go( )*
wr	*Angular speed of the robot*	*go( )*
speed1	*Speed of Motor 1*	*go( )*
speed2	*Speed of Motor 2*	*go( )*
speed3	*Speed of Motor 3*	*go( )*
INA_MOT[4]	*Motors INA pins*	*init_motori( )*
INB_MOT[4]	*Motors INB pins*	*init_motori( )*
PWM_MOT[4]	*Motors PWM Pins*	*init_motori( )*

INA	*INB*	*Responsible of the motor movements*
1	*0*	*Clockwise*
0	*1*	*Anti-clockwise*

TAB "ATTACCANTE" (Forward Strategy)

Functions				
NAME	**PARAMETERS**	**CALLED FUNCTIONS**	**PURPOSE**	**OUTPUT**
void attaccante( )	*sensore,palla_sensore,direzione, sonar_dx,sonar_sx, G_direzione[sensore], VEL_ATT[VELxZONE( )], gradi_norm,letto,time_testugine, piezo,bip,velocita.*	*VELxZONE(), leggi_ultrasuoni( ), go( )*	*Move the robot to the ball and to the opponent goal*	-
byte VELxZONE( )	*ZONE_*,FASCIA_*,v,MED,MAX*	*interpola_pos( )*	*Outputs the best speed for the robot guessing where it is*	*v*

Variables		
NAME	**PURPOSE**	**USED IN**
bip	*Time of buzzer beep*	*void attaccante( )*
sensore	*Number (0-19) of the sensor best "seeing" the ball*	*void attaccante( )*
velocita	*Current robot speed*	*void attaccante( )*
G_direzione[20]	*Array of angles for the robot movements for each ball sensor*	*void attaccante( )*
direzione	*Direction of the robot movement*	*void attaccante( )*
gradi_norm	*Angle between the robot and the opponent goal (Computing it as 0°)*	*void attaccante( )*
VEL_ATT	*Attack speed*	*void attaccante( )*
sonar_dx	*Reading from the right sonar.*	*void attaccante( )*
sonar_sx	*Reading from the left sonar.*	*void attaccante( )*
MED	*Medium speed*	*byte VELxZONE( )*
MAX	*Max speed*	*byte VELxZONE( )*
*ZONE_**	*Contiene la posizione calcolata del robot nel campo tra le 9 zone. NORD,NORD_EST,NORD_OVEST,CENTRO,EST,OVEST,SUD, SUD_OVEST,SUD_EST.*	*byte VELxZONE( )*
*FASCIA_**	*Numbers to add to the probability matrix depending on the evaluated robot position*	*byte VELxZONE( )*
v	*Robot speed*	*byte VELxZONE( )*
piezo	*Buzzer Pin*	*void attaccante( )*
palla_sensore	*Number (0-19) of the sensor best "seeing" the ball. Its value is the same as sensor*	*void attaccante( )*

TAB "PORTIERE" (Goal keeper strategy)

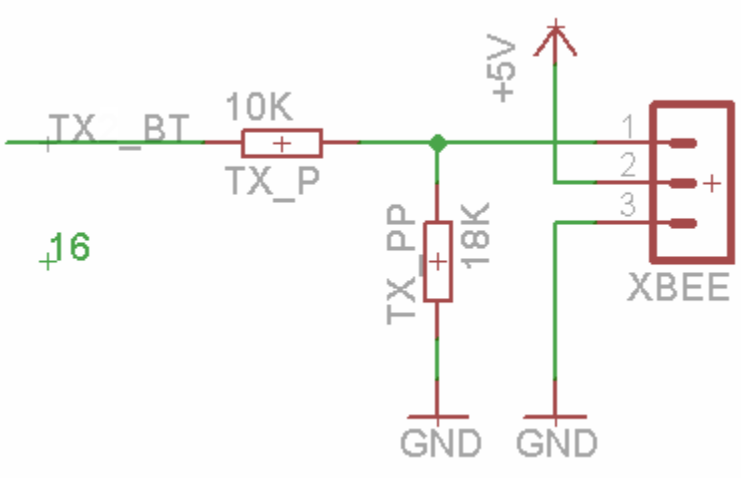
Functions				
NAME	**PARAMETERS**	**CALLED FUNCTIONS**	**PURPOSE**	**OUTPUT**
portiere_improved()	-	*interpola_pos(),attaccante(),ritorna_fascia_sud(), go(), ritorna_porta()*	*Makes the robot stay near its gool, tracking the ball. The goal keeper attacks only if the ball is very near*	-

Variables		
NAME	**PURPOSE**	**USED IN**
time_attacco		*portiere_improved()*
posizione	*Robot Position*	*portiere_improved()*
fascia_sud	*True = Robot in the South Zone (near its goal)*	*portiere_improved()*
palla_vista	*True = The robot sees the ball*	*portiere_improved()*
palla_sensore	*Number (0-19) of the sensor best "seeing" the ball.*	*portiere_improved()*
*VEL_**	*Speeds Array*	*portiere_improved()*
gradi_norm	*Degree of the opponent goal versus the real North*	*portiere_improved()*
tempo_s_palla	*Time between change of sensor ball*	*portiere_improved()*
*sonar_**	*Reading from sonars*	*portiere_improved()*

DEBUG STATION

To be able to track and check the robot behaviour our team inherited and improved a software in Processing language called "Debug Station".

The debug station runs on the PC and communicates with the robot main board via WI-FI using the serial n. 3 of the microcontroller.



It is possible to monitor remotely any sensor without interfering with the main program running inside the robot, so we can analyse the robot while it is playing.
(of course not during the official games!)
This it is possible because the stream of data is packed accurately and the transmission occurs at regular intervals of 4 ms.
If the robot is not playing, the transmission occurrence is increased at 2ms.
Via a menu the user of the debug station can select the area he is interested in and change it with a click of the mouse.