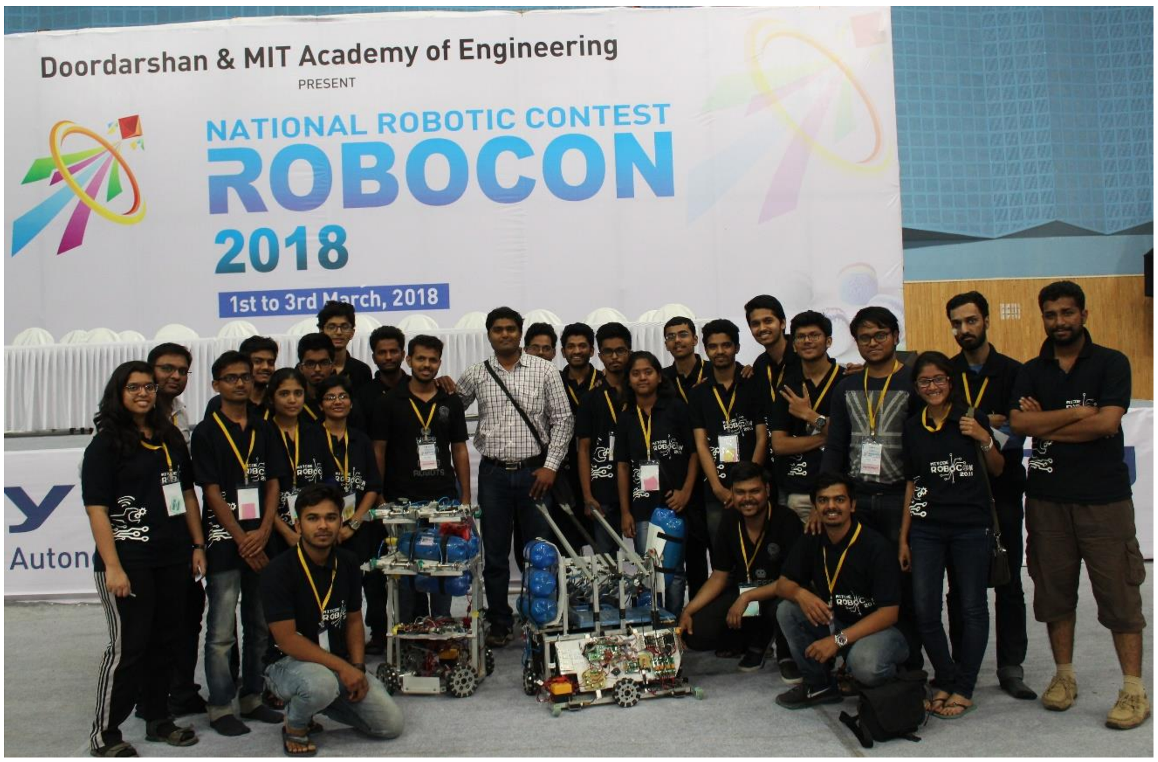
TEAM ROBOCON ANNUAL REPORT

A comprehensive documentation of development process.



Declaration

I hereby declare that the project work entitled as 'Team Robocon Annual Report: A comprehensive documentation of the development process' is an authentic record of the work done by the Robocon team of MIT college of engineering as required for the national Robocon 2018 with the due support of MIT College of Engineering and its Faculty during August 2018 and March 2018.

15th March, 2018

Team Robocon

Prof. G. G. Narkhede
ROBOCON Coordinator
ENTC

Prof. R. S. Bobade
ROBOCON Coordinator
Mechanical

Prof. M. R. Saraf
ROBOCON Coordinator
Mechanical

Prof. Dr. S. B. Barve
ROBOCON Incharge
MITCOE

Prof. Dr. A. S. Hiwale
Principal MITCOE

Prof. Dr. R. V. Pujeri
Director, MITCOE.

Acknowledgement

We would like to express our gratitude to all those who gave us the possibility to complete this very project. We want to thank all the engineering departments of the esteemed institution MIT College of Engineering for giving us such a golden opportunity to commence this project in the first instance. We have to furthermore thank all the teachers, heads of departments and our Director Prof. Dr. R V Pujeri sir who encouraged us to go ahead with this project. I am also thankful to the workshop department of MITCOE for their endless support.

We are deeply indebted to our Prof. G.G. Narkhede Sir of the ENTC department whose help, stimulating suggestions and encouragement helped us in all the time during the project tenure. We are also thankful to our Robocon Coordinator Prof. Mangesh Saraf Sir for helping us during the project.

Our colleagues from all the disciplines of engineering worked unstintingly throughout the development process. We would like to appreciate the determination and hard work of each and every team member whose tolerance, unending help and valuable hints and ideas helped the team as a whole maintain its integrity and share a common goal.

Especially, we would like to give a very special thanks to our seniors and alumni who enabled us to complete this work and being a constant source of enthusiasm and motivation.

And last but not the least, we would like to thank god for the successful completion of the project.

Abstract

This project was done according to the theme and rules as proposed by the Asia Broadcasting Union (ABU) and the Robocon committee of the host country Vietnam for the year 2018.

The theme names “NINH BINH: The Flying Dragon” was set by Vietnam in this year. According to theme we had to make two robots that would shoot shuttlecock. One was manual and one was automatic. Manual Robot had to pick the shuttlecock from the rack and transfer it to Automatic Robot for this one point was awarded. Then Automatic Robot will shoot that from predefined Throwing zones, i.e. TZ1, TZ2, and TZ3. In front of each zone there were vertical rings, the shuttlecock was supposed to go through that ring. Each zone had points 10, 15 and 30 respectively. If Robots were able to complete all 3 zones and the shuttlecock has landed in the bowl in front of ring, then it would flash win called “Rong Bay”.

The bot was made by broad use of 12mm and 19mm Square aluminum sections with 1 and 2mm thickness as per requirement. Also, there was a wide use of laser cut and bent aluminum and stainless steel fixtures. Omni wheel were used for support. Pneumatic systems were also a major part of the design engineering. Pneumatic pistons were used for achieving the desired actuation of shooting disc. Direction Control Valves or DCV’s and Solenoid valves were used to achieve desired actuations of the pistons. The various joining processes used were Nuts and Bolts, Aluminum TIG welding, Soldering, Epoxy resin and press fitting. FCVs were used to control speed of actuation.

Various Electronic Devices used included Arduino Board, SaberTooth (Motor Driver), DC Motors like Cytron, 24 Volt e-bike Motors. We made our own motor driver to get the flexibility that we needed. To get the feedback of bot’s status we used Tachometer (to get the RPM of shooting wheel), Pot (to get angle of shooting plane) and Encoders (to implement PID).

Main challenge was Automation, once the match starts we cannot touch or communicate with automatic robot. There were lines to guide Automatic Robot to throwing zones. Instead of choosing LSA-08 a line sensor by Cytron, we opted to use camera and do image processing to extract data about position and location of Robot.

The programming concepts used were PID i.e. proportional Differential Integral for controlling the speeds of the two drive motors based on the encoder values. The control of bot was achieved using advanced programming techniques.

INDEX

MECHANICAL
Autonomous Robot

Shooting Mechanism

1. Motor Powered Shooting Mechanism

- Need
  - To get the precise trajectory of shuttles through rings mounted on poles. To select parameters in a way to achieve all three targets and successfully launch shuttle in Rong Bay bowl.
- Materials used
  - Aluminium square tubes of 19 and 12 mm side length with 2 and 1 mm thickness respectively.
  - Aluminium plates of 2 mm thickness.
  - ABS.
- Components used
  - Dc Motor
  - D printed hubs
- Testing
  - In this mechanism arm length is fixed at 300mm and shooting trials were taken by varying delay and speed of the motor.
  - Trials were taken for all three positions i.e. TZ1, TZ2, and TZ3.
  - For few trials Cytron dc motor is used but due inefficiency of motor power achieving Rong Bay was a problem. Hence we used Maxon dc motor we was much precise and faster than Cytron DC motor.
- Advantages
  - Single throwing arm is required.
  - Precise and reliable shooting.
- Disadvantages
  - Heating of dc motor after few trials.
  - Difficulty in achieving Rong Bay.

2. Piston Mechanism

- Need
    - To make more reliable mechanism and to achieve Rong Bay in minimum possible time.

- Materials used
    - Aluminium square tubes of side length 19mm and 2 mm in thickness for frame and throwing arm.
    - Carbon fibre square tube of 20 mm side length and 1mm thickness for throwing arm.
    - SS 304 rods of 10mm and 8mm dia.

- Components used
    - Pneumatic pistons with 160mm stroke length and 20 mm bore diameter for throwing arm and pneumatics piston 100 mm and 16mm bore diameter for C-mechanism.
    - Direct control valves, flow control valves, analog and digital pressure gauges, pneumatic bottles.
    - Corrugated sheet of 3mm thickness.

- Testing
    - We tested this mechanism for all three positions i.e. TZ1, TZ2, TZ and for Rong Bay.
    - After few testing we decide to go with pneumatic pistons of 160 mm stroke length and 20 mm bore diameter for TZ3 and same pistons for TZ1 and TZ2 with flow control valves attached.
    - Due to bending problems in throwing arms of TZ3, we used Carbon fibre square tubes as the throwing arm.
    - We used SS rods as a supporting member at the pivots of piston and throwing arm.
    - Laser cut notches are adding at the end of the throwing arm for precise communication between Manual and Autonomous robot of shuttles.

- C-mechanism for shuttle holding during manoeuvring for better transfer of shuttle. Pneumatic pistons of 100mm stroke length and 16mm bore diameter are used here.

- Advantages
  - More precise mechanism for achieving Rong Bay.
  - Accomplishing all shooting targets minimum possible time.
  - Communication time between Manual and Autonomous robot is reduced.
  - Due to multiple shooting arms we gained strategic advantage.

Base Frame Drive
Omni Wheels Plus Drive

- Need
  - Easy and precise manoeuvring on plane field to follow white lines to reach throwing area.
- Materials used
  - Aluminium square pipes of side length 19mm and 2mm thickness.
  - Omni wheels of 127mm diameter.
  - Cytron high torque motors.
  - Aluminium front and back mounts for Cytron motor.
- Testing
  - Omni drive are used in plus format due ease in manoeuvring.
  - Due to bending in outer base frame, a new compact design is adopted.
  - Space is given in the centre of the frame for field of view of camera.
  - Cytron dc motors are used with aluminium motor mounts and aluminium back mounts.
- Advantages
  - Easy to manufacture and better performance with image processing.
  - We were able to achieve high speeds using this drive.
  - Due to symmetric design centre of gravity is near to central axis giving an upper hand while coding for manoeuvring.

Manual Robot
Gripping Mechanism

- Need -
    - To grip shuttlecocks arranged on rack and to release them to autonomous robot.
- Materials used-
    - Nylon hubs manufactured on lathe machine
    - Laser cut aluminum plates
    - Stainless steel piston mounts
- Components used-
    - Double actuating pistons with 8cm stroke length
    - DCV
- Testing-
    - Trials were taken for all three positions i.e. TZ1, TZ2, and TZ3
    - It worked accurately with instantaneous piston movement.
- Advantages-
    - Single DCV is required for gripping one shuttlecock.
    - Precise and reliable gripping.
- Disadvantages-
    - Air leakage from connections must be avoided.

Base Frame Drive
Omni Wheel Drive

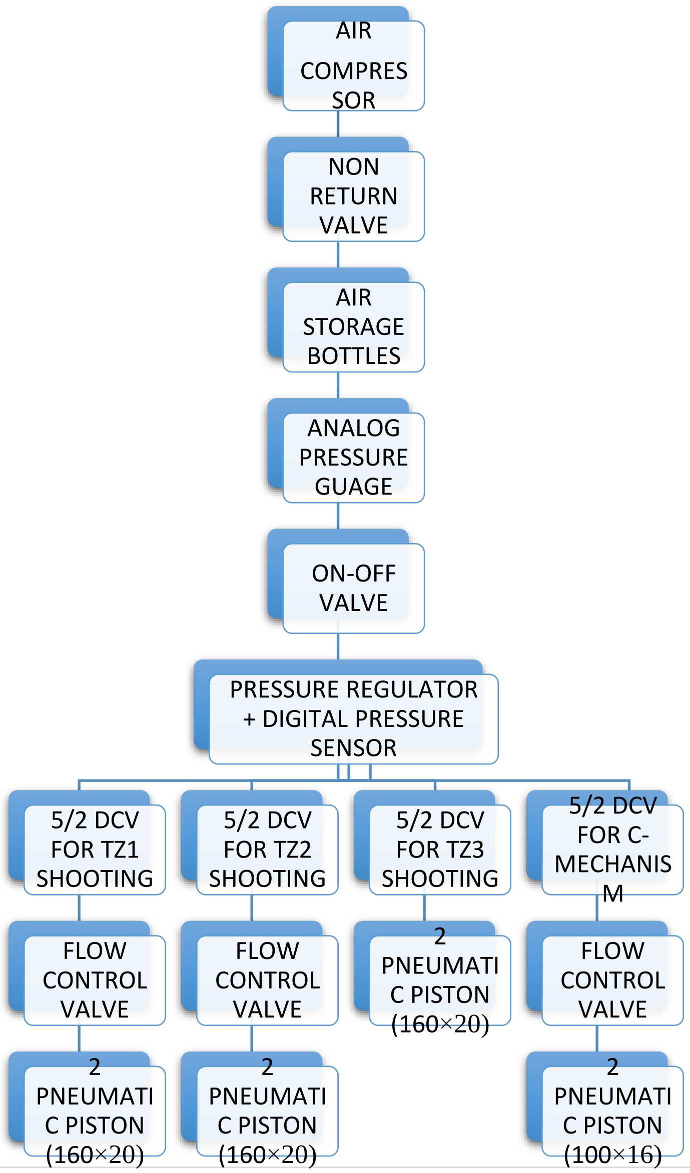
- Need-
  - Easy and precise maneuvering on plane field to reach autonomous robot.
- Materials-
  - Aluminum square pipes of side length 19mm and 2mm thickness.
  - Omni wheels of 152mm diameter.
  - Aluminum front and back mounts for Cytron motor manufactured on CNC lathe.
- Testing-
  - Omni drive are used in X format due ease in maneuvering.
  - Due to bending in outer base frame, parallel sections supporting back mounts of motor are added.
  - Cytron dc motors are used with aluminum motor mounts and aluminum back mounts.
- Advantages-
  - Easy to manufacture and fast maneuvering.
  - We were able to achieve high speeds using this drive.
- Disadvantages-
  - Difficult to manufacture base frame.

Pneumatic System

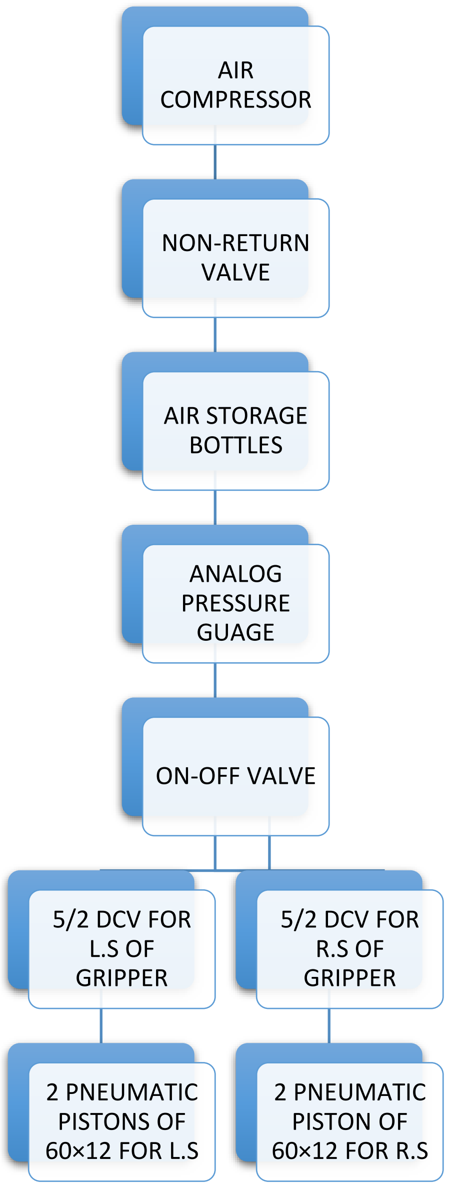
Introduction

Pneumatic systems form the most primitive and distinct class of mechanical control engineering. They are classified under the term 'Fluid Power Control', so pneumatic system was major part of design. Pneumatic pistons were used for achieving desired actuation of several mechanisms. Direction Control Valves or DCV's and Solenoid valves were used to achieve desired actuations of the pistons.

Pneumatic Control Flow Chart for Automatic Robot



Pneumatic Control Flow Chart for Manual Robot


AIR COMPRESSOR
NON-RETURN VALVE
AIR STORAGE BOTTLES
ANALOG PRESSURE GUAGE
ON-OFF VALVE
5/2 DCV FOR L.S OF GRIPPER
5/2 DCV FOR R.S OF GRIPPER
2 PNEUMATIC PISTONS OF 60×12 FOR L.S
2 PNEUMATIC PISTON OF 60×12 FOR R.S

- Components used in system

  1) Pneumatic Pistons
  2) Pneumatic Wires (6 OD)
  3) Inline Flow Control Valve
  4) Piston Flow Control
  5) Pressure Regulator
  6) Pressure Gauge
  7) Digital Pressure Sensor
  8) 5/2 DCV (6)
  9) T Connectors (6 OD)
  10) Y Union (6 OD)
  11) Cross Union
  12) Male Connectors of diameter 6× 1/8
  13) Female Connectors of diameter 6× 1/8
  14) Non-Return Valve and ON-OFF Valve
  15) Rod and Spherical Eye

- Pneumatic pistons

  Piston and arms are attached as reciprocating cylinder mechanism to throw a shuttle attached in the laser cut notch using knot at the tail of shuttle. The mechanism was equipped with pre-calibrated cylinders of stroke length 160mm and bore diameter 20mm and this combination was sufficient to achieve Rong Bay and all three shootings. For TZ1 and TZ2 pistons, Inline flow control valves and piston flow control valves were used to get desired distances. Output pressure was pre-set to 3.78 Bar for all shootings and distances were achieved by controlling flow of inline flow control valve.

- Flow control valve

  We know the concepts of meter in and meter out in pneumatic circuits. For shooting mechanism, backward stroke must be faster in order to achieve specific distances. Hence to control velocity of pistons FCV's were used. They were used to get desired output.

- Pressure Regulator

  As we know every shuttle has to be pushed with same force as we need similar trajectory.

  As $F = P * A =$ (Pressure * area)

  as long as pressure is same we'll get same force. To keep pressure constant, pressure regulator was used. Even though system pressure was 6 bar (allowed and pressure in air bottles), output was set to 3.78 bar (which was set according to trials for better landing).

- Digital pressure sensor

  To overcome the errors and difficulties occurred in setting precise pressure in regulator digital pressure sensor was used. To achieve the required trajectory, the pressure was to be set with great accuracy; this precision couldn't be obtained with analog pressure gauge. Hence, a digital pressure sensor with the least count of 0.01 was used to set the pressure.

- Air bottles

  Total 7 bottles were used on the automatic bot and 4 on manual bot. Each bottle having 3l volume giving us 21L volume for AR and 12L for MR to save the AIR .NO of bottles were calculated for actuation of 6 pistons for 14 actuations (for 9 Shuttles). For greater pressure handling of bottles silicon sealants and dentistry solution were used.

Rack

- Need:

MR should be capable of loading shuttles in minimum time. And to hold as many shuttles as possible.

- Materials required:

1- Aluminium square tube with side length 19mm and thickness 2mm.

2- Laser cut notches and 2-D L brackets.

- Testing:

Rack was tested to see its holding strength while MR is clamping shuttles from Rack notches to load it. It was tested in such a way that even if MR collides with the Rack, shuttle would not fall.

- Information:

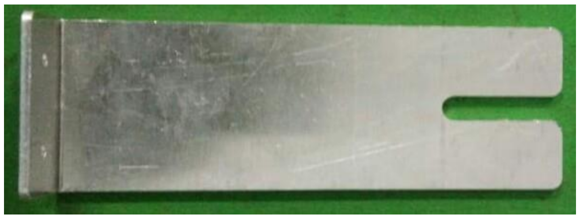
Rack was approx. 1400 mm in length and 1000 mm in height and was within dimension limit. It was made up of two horizontal as well as vertical sections of Aluminium square pipe. There were a few support sections at the bottom to make the base of the rack. The notches were riveted to the two horizontal sections at 220 mm consecutively and the distance between two horizontal sections was 240 mm to reduce the degree of freedom of shuttles and hold it still in notches. The notches 120mm in length,40mm in breadth and 2 mm in thickness. They were Laser cuts and then bended at 20mm distance opposite to the U shape cut.

- Advantages:

• Ease in loading for MR.

• Loading time is reduced.

• Stable and Simple Design.



ELECTRONICS AND TELECOMMUNICATION

Introduction

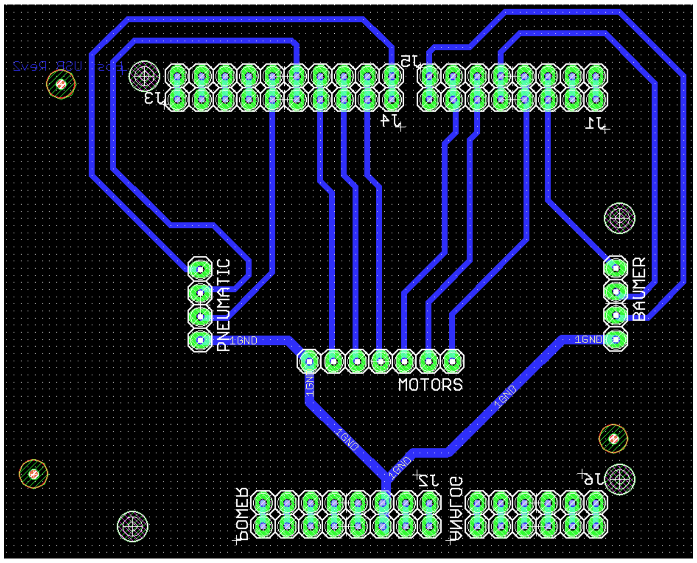
This year, the task required us to throw the Vietnamese shuttlecock through the loops present at certain heights. The team took part in testing various mechanisms, and hence selected the most efficient one for further prototyping and testing.

After the mechanism was finalized by the all departments, the machine was sent to E&TC for finalizing the boards to be present on the Automatic and Manual Robots. With the aid of the Computer department, the requirements for each robot was noted down and the work progressed there-on.

Autonomous Robot

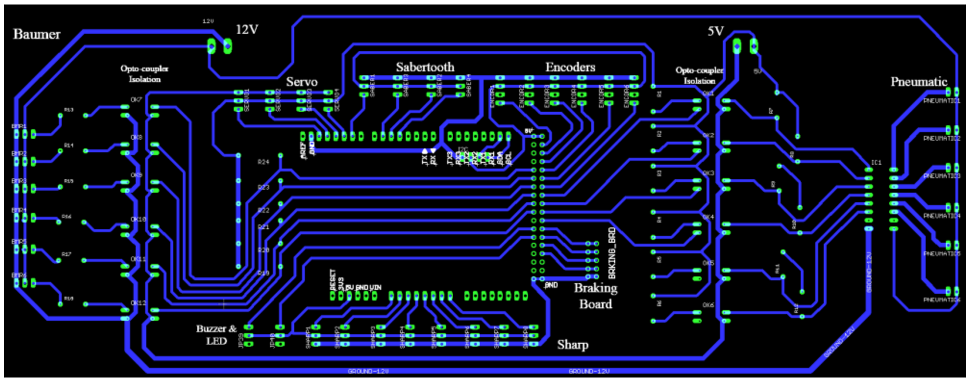
We used an Arduino UNO in the initial stage as the machine was in the initial development stage so the idea was to get familiar with the components in Autodesk EAGLE for creating the design of the boards. Before the boards were printed, jumper wires were used to wire the interface on AR.

Initially the requirements for the AR shield were 4 motors via 2x25 Sabretooth Motor Driver; 3 Baumer Sensors; and 3 Pneumatic actuators. The initial AR shield just contained the pinouts for the respective sensors/actuators so as to have easier debugging.



Later, we shifted to Arduino MEGA 2560, as we needed more number of pins.

However, on this we decided to integrate all the systems on one board, i.e. the Baumer, pneumatic, sharp, servo and the Sabertooth signals on a single PCB. Basically, we implemented a Distributed control system, which is easier for debugging and troubleshooting problems.



Baumer sensor sends a digital signal of 12V which was dialled down by a 1k Ohm current limiting resistor. Thus, turning on the LED in the opto coupler, which results in the photo transistor behaving as a closed switch, which sends a signal to the MEGA, making the respective pin Logic HIGH.

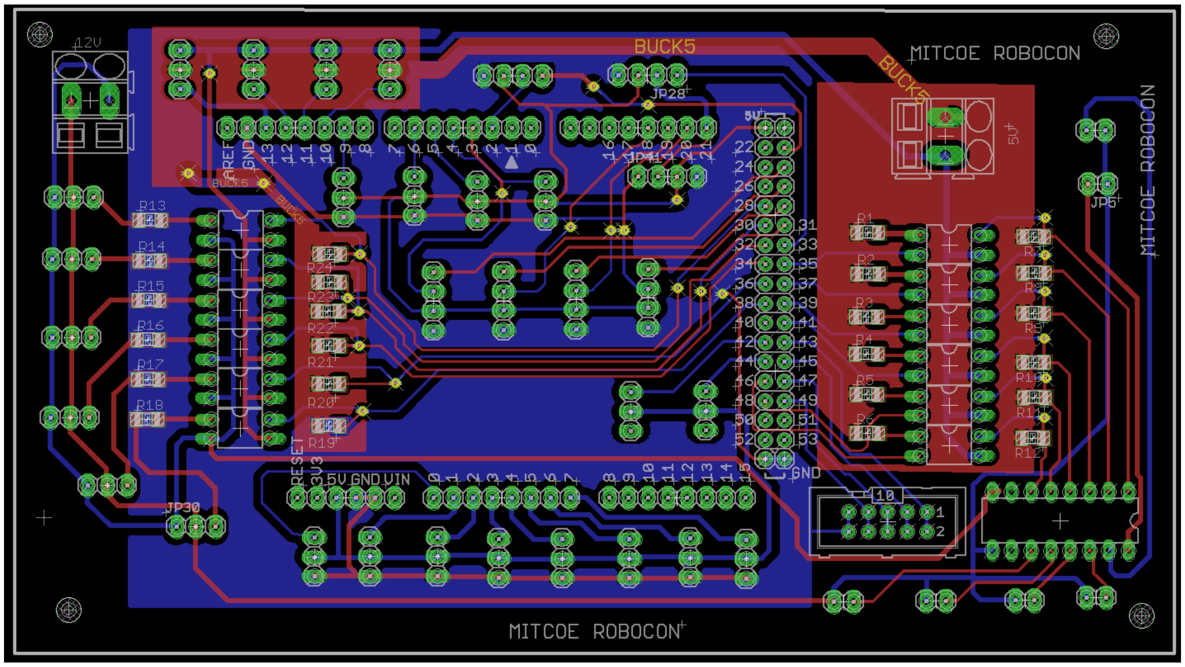
Depending upon which Baumer was sending the signal, the respective MEGA pin for the Pneumatic actuator circuit is made Logic LOW (as the "current sink" logic is used). Again a Pull up resistor is present between the Mega and the opto-coupler LED. Opto coupler provides isolation to the controller, separating grounds of each of the circuits.

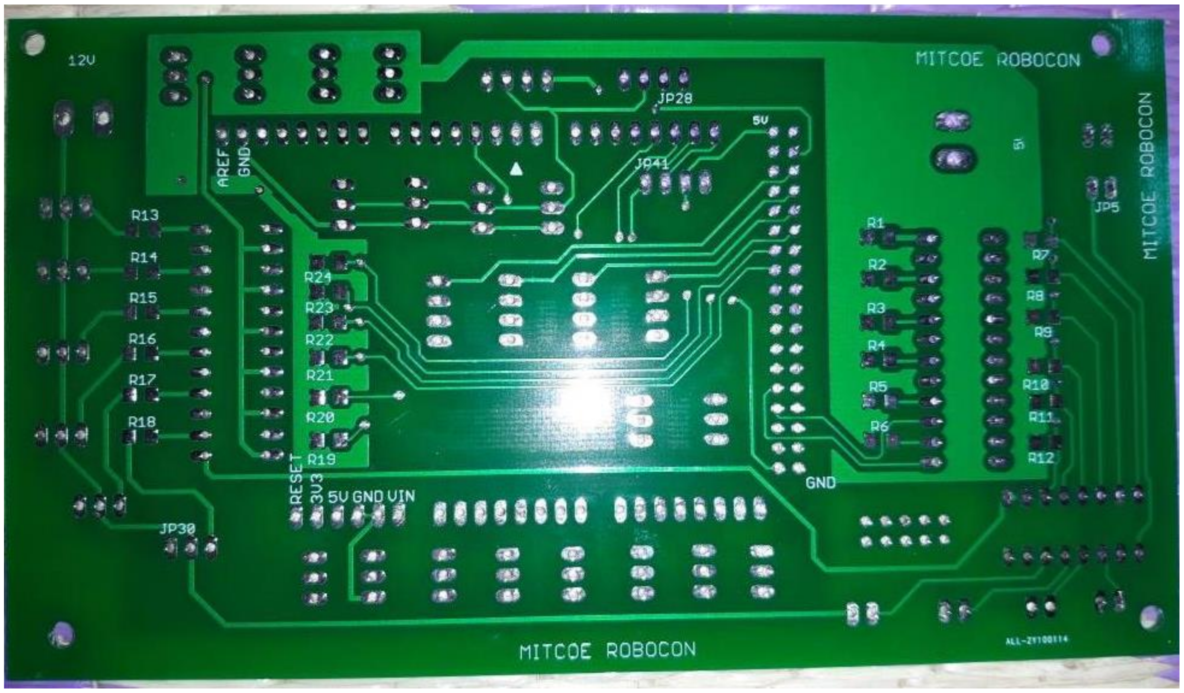
Servo motors were initially included to help provide stability to the shuttlecocks passed on from the Manual Robots. However, it was then decided to shift onto pistons for quicker action.

Sabertooth signals having connection in the form of 3 pin relimates (GND, Vcc, Signal). Encoders were used to get the exact RPMs of the motors.

Sharp sensors which give an analog output were used to measure distances from the boundary walls present on the field. They also had a similar, 3 pin relimate connection (GND, Vcc, Signal). The braking board uses an FRC Connection to send signals to respective motors for braking action.

Terminal blocks were used for external power supply. I2C connections were present for master-slave communications. Wiring in general, was dramatically reduced after the shield was implemented. Later, a double-sided PCB was printed to make it more compact and efficient.





Manual Robot

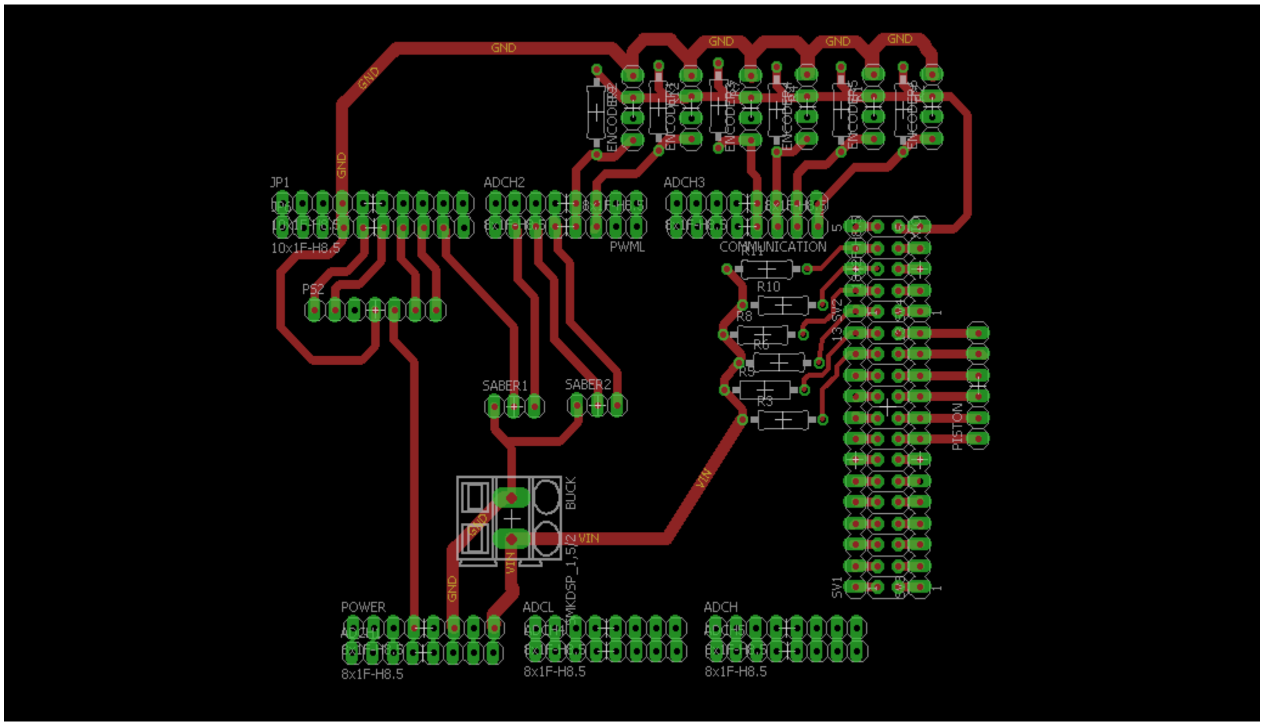
The main aim was to build a control system for a manual robot. The controller used for that was arduino mega 2560.

Need:

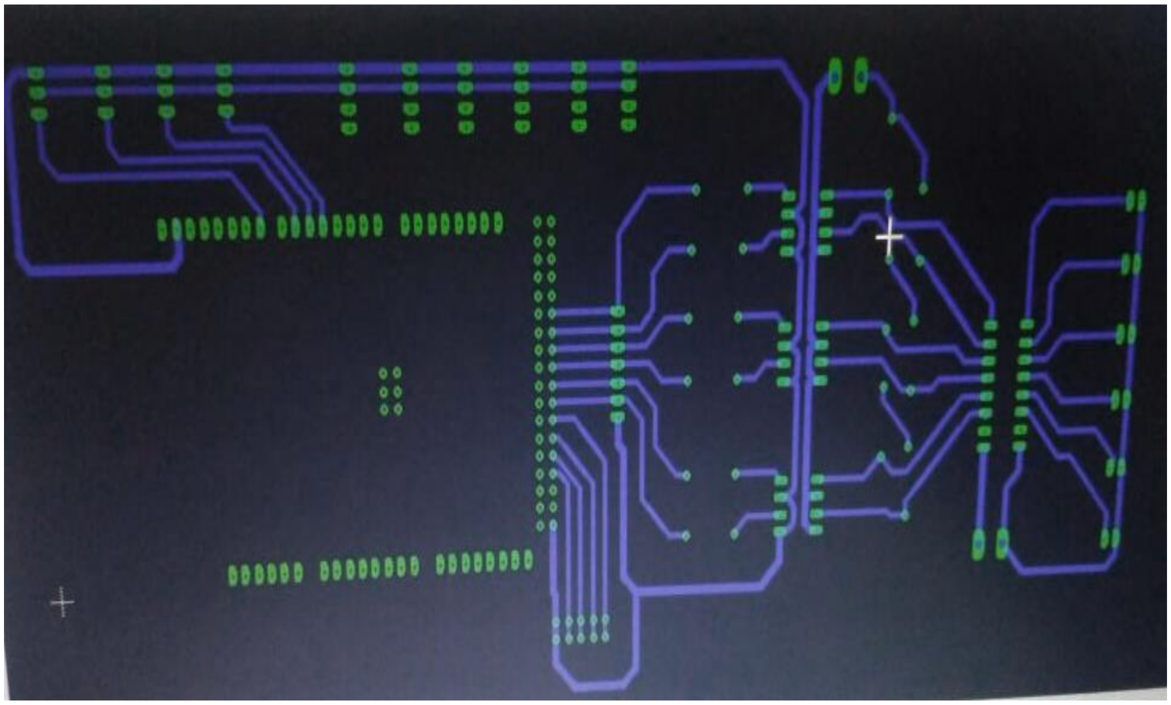
The shield was needed for distributing the controls of the arduino mega. That in turn reduced our wiring efforts and also reduced the other PCBs that we might have to manufacture if there isn't a mega shield.

Design:

This shield was designed such that it will directly fit upon the mega.so the dimensions were taken into consideration. The PCB that we have built was single sided PCB. All the PCBs are designed using eagle that is a professional software for designing the PCBs. the connection of the new and the old MR shield is the same but the new one is 2 sided PCB.



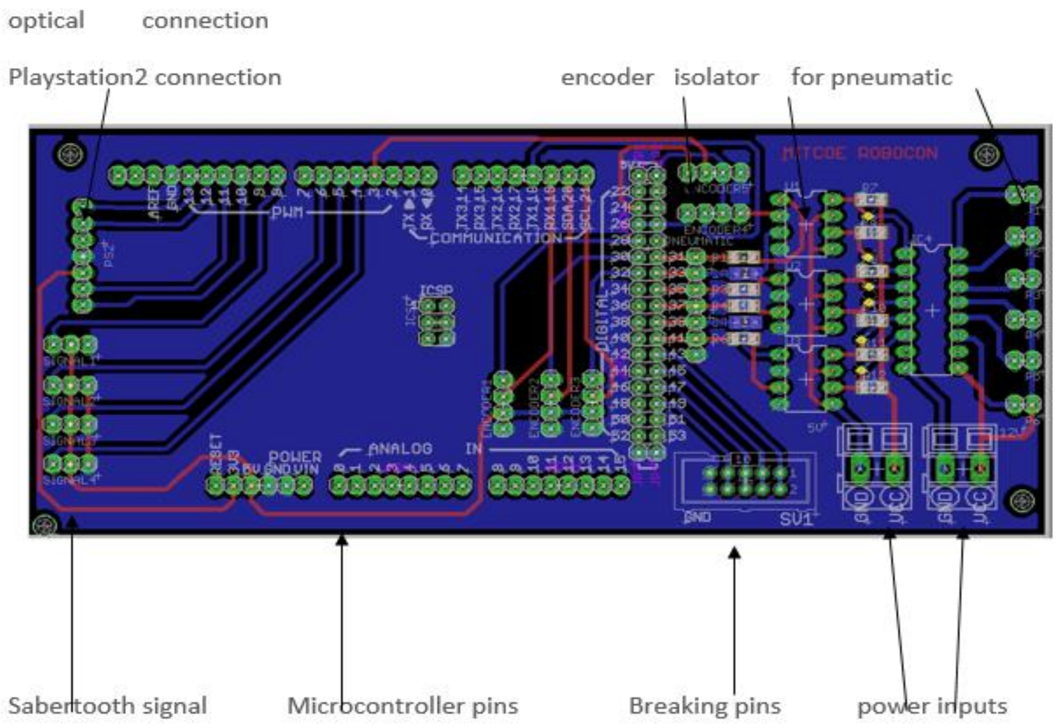
Modified MR Shield:



New Manual Robot Shield:

The new MR shield is a distributed control system. In which we embedded different control circuits. The main purpose of the MR SHIELD was to connect the required pins of the mega to the peripherals. The shield also had the pneumatic circuit which was used to actuate the DCV's. The PCB were designed double sided. The shield reduced the wiring efforts by bringing all the pins of the mega on the single board. MR shield is nothing but development board of Arduino Mega 2560. The board also used the optical isolator circuit for isolating the peripheral ground and the microcontroller ground.

The Design:



Design of the shield consists of the pins for encoder of the Dc motor, Pneumatics, Breaking board, ps2, Sabertooth signal. The shield also consists of the IC ULN2003 which is a Darlington pair.

- The encoders pins for the motor are power and one digital and one interrupt pin. A non-quadrature incremental encoder provides a single pulse to the controller for every incremental motion of the motor shaft. A quadrature encoder provides two pulses, out of phase, that can be used to detect direction also. We have used motors with quadrature encoders.
- For the actuation of the solenoids that in turn actuates the DCVs, we have used pneumatic circuit board. The board consists of "ULN2003"and optical isolator that isolates the signal from the actual signal given to the IC.
- The motor driver that we have used was Sabertooth which comes with different operating modes. The maximum current rating for the motor driver on the manual robot is 25A. For driving the motors there is a

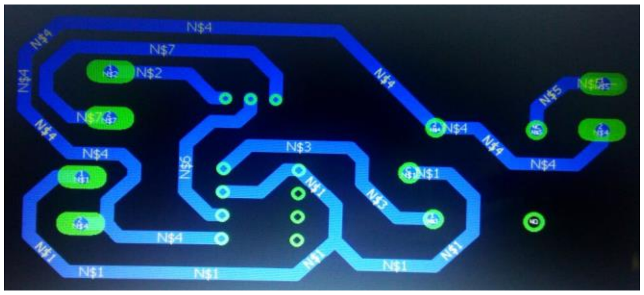
connection on the board with power signals and one digital pin for each of the motor.

- The board has power input of 5V and 12V for microcontroller operation and pneumatic operation respectively.

Important Circuits

Sabertooth Under voltage Protection Circuit

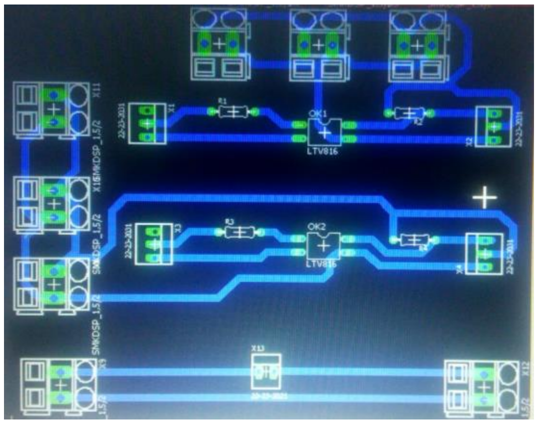
The main purpose of using this circuit was to avoid the Sabertooth from getting less voltage. Because it was seen that when the Sabertooth was given less supply voltage, it provided less current to the motors. This circuit also provides battery over discharge protection. This circuit consist of a comparator with two input voltage, out of which one was acting as a reference voltage. The reference voltage was set with the help of variable resistance. Relay was used at output for switching action. The 12V input from the battery was given to the inverting input of comparator. Another 12V input, dropping down to 10V with the help of potentiometer was provided as input to non- inverting terminal of comparator. Initially, the voltage at inverting input of was higher than non- inverting, so the output of the comparator is negative saturation voltage and hence relay will remain off keeping it in NC position and passing the input 12V at the output to Sabertooth. As the voltage at inverting drops down, and once it becomes less than 10V the output of the comparator will switch to positive saturation. This output of comparator is given to the relay coil, which will switch and make connection with NO and disconnects the output.



LED, Buzzer and Distribution Board:

In the rulebook, it was mentioned that a part of automatic robot must indicate a sign such as flashing light when the throwing action starts.

This board includes led and buzzer connection which was isolated from the shield with the help of opto-coupler. LED and buzzer were given 5V and 12V respectively. Here, the signal for blinking of led was send from AR shield. After the sinking of signal at input, the LED glows at the output.



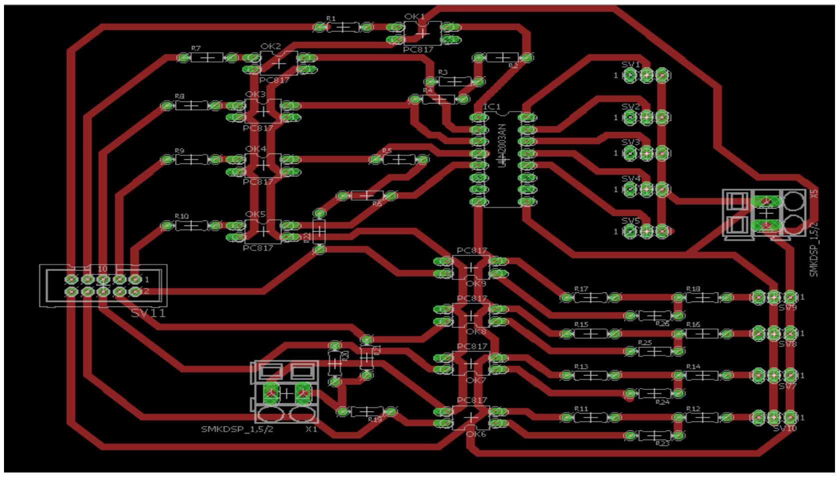
Sabertooth Protection Circuit:

This circuit is providing the protection to signal input terminals (S1 & S2) of the Sabertooth by providing an isolation between input and output using opto-coupler. Isolation between input and output is one of the best protection method. Because any problem on either side will keep one of the side safe. The input signal to the opto-coupler is given from AR shield. A separate 5V supply is given to the collector of the transistor present inside the opto-coupler through a pull-up resistor. The voltage drop across the transistor is taken as output signal and is further given to signal input terminals of Sabertooth.

Pneumatic Circuit:

This circuit will actuate the DCV, whenever the solenoid associated with it will get supply of 12V.This board is a Pneumatic solenoid actuation circuit. The input signal is given from the AR shield. The signal is given to opto-coupler. The signal of 5V from the collector of the opto-coupler is given as input to IC ULN2003. The ULN2003 is high voltage, high current Darlington transistor array. IC consist of seven NPN Darlington pairs that features a high voltage output. By default, output of every pin of ULN2003 gives 12V. When the input signal is detected, the corresponding output pin will be grounded, and the loop will get completed leading to actuation of pneumatic solenoid.

Pneumatic with Baumer circuit:



Communication on the bots:

The primary task was to research on the most ideal and efficient communication to be implemented on the robots.

Choosing between I2C and SPI, the two main serial communication options, can be quite a challenge and have a significant impact on the design of a project, especially if the wrong communication protocol is used. Both SPI and I2C bring their own advantages and limitations as communication protocols which make them each suited for specific applications.

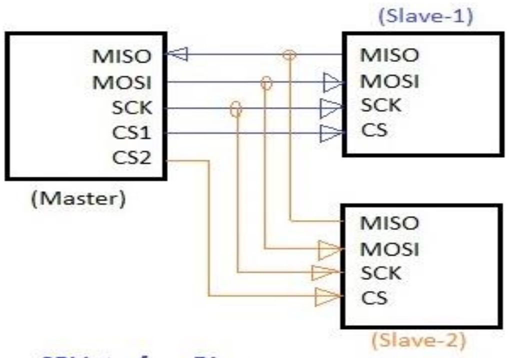
SPI:

SPI, or Serial to Peripheral Interface, is a very low power, four wire serial communication interface designed for IC controllers and peripherals to communicate with each other.

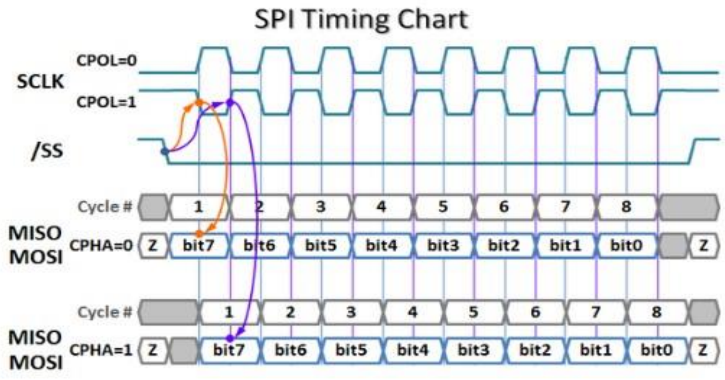
The SPI bus is a full-duplex bus, which allows communication to flow to and from the master device simultaneously at rates of up to 10Mbps. The high-speed operation of SPI generally limits it from being used to communicate between components on separate PCBs due to the increase in capacitance that longer distance communication adds to the signal lines. PCB capacitance can also limit the length of SPI communication lines.

While SPI is an established protocol, it is not an official standard which leads to several variants and SPI customizations which can lead to compatibility issues. SPI implementations were checked between master controllers and slave peripherals to ensure that the combination will not have any unexpected

l impact the development of a product



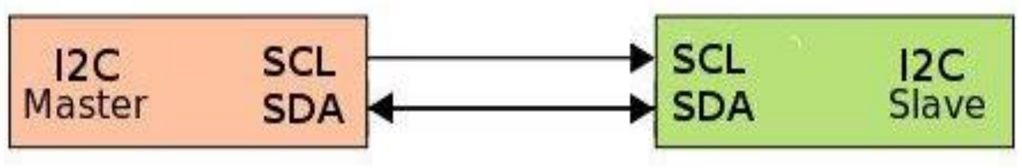
SPI Interface Diagram



I2C:

I2C is an official standard serial communication protocol that only requires two signal lines that was designed for communication between chips on a PCB.

I2C was originally designed for 100kbps communication but faster data transmission modes have been developed over the years to achieve speeds of up to 3.4Mbps. The I2C protocol has been established as an official standard, which provides for good compatibility among I2C implementations and good backward compatibility.



Selecting Between I2C and SPI:

Each communication protocol will have distinct advantages which will tend to distinguish itself as it applies to your application. The key distinctions between I2C and SPI are:

- I2C requires only two wires, while SPI requires three or four
- SPI supports higher speed full-duplex communication while I2C is slower
- I2C draws more power than SPI.

- I2C supports multiple devices on the same bus without additional select signal lines through in-communication device addressing while SPI requires additional signal lines to manage multiple devices on the same bus
- I2C ensures that data sent is received by the slave device while SPI does not verify that data is received correctly
- I2C can be locked up by one device that fails to release the communication bus
- SPI cannot transmit off the PCB while I2C can, albeit at low data transmission speeds
- I2C is cheaper to implement than the SPI communication protocol
- SPI only supports one master device on the bus while I2C supports multiple master devices
- I2C is less susceptible to noise than SPI
- SPI can only travel short distances and rarely off of the PCB while I2C can transmit data over much greater distances, although at low data rates
- The lack of a formal standard has resulted in several variations of the SPI protocol, variations which have been largely avoided with the I2C protocol

Both SPI and I2C are good communication options, but each has a few distinct advantage and preferred applications. Overall, SPI is better for high speed and low power applications while I2C is better for suited for communication with a large number of peripherals and dynamic changing of the master device role among the peripherals on the I2C bus.

Both SPI and I2C are robust, stable communication protocols for embedded applications that were implemented on our robots.

Our Decision:

As the Automatic Robot had a lot of peripherals as Baumer, sharp circuit, pneumatic circuit etc. we implemented I2C communication.

The Manual Robot has one slave i.e. the XBOX controller which was controlled via USB host shield and therefore the mode of communication was SPI.

Sensors

BAUMER: O500.GR-11096062

- Proximity Sensors with inbuilt Processor.
- Gives Digital Output depending whether obstacle is detected or not.

General data:

Type	background suppression
light source	pulsed red LED
sensing distance Tw	60 ... 550 mm
sensing range Tb	30 ... 600 mm
power on indication	LED green
light indicator	LED yellow
sensing distance adjustment	qTeach
wave length	630 nm
suppression of reciprocal influence	Yes
alignment optical axis	< 1°

electrical data:

response time / release time	< 1 ms
voltage supply range +Vs	10 ... 30 VDC
current consumption max. (no load)	40 mA
current consumption type	30 mA
voltage drop Vd	< 3 VDC
output function	light / dark operate
output circuit	push-pull
output current	< 100 mA
short circuit protection	Yes
reverse polarity protection	Yes

Use:

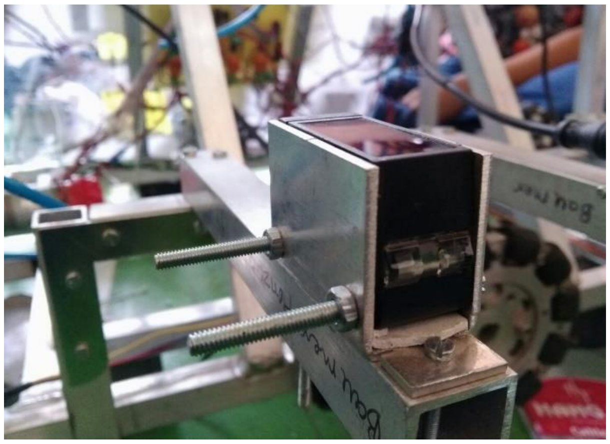
- Three Baumers were mounted on the sections of Automatic bot right below the arms. They were used for detecting the shuttlecocks which were attached on the three arms of Automatic Robot.
- One Baumer which was on the back side of the Automatic bot was used to detect the presence of Manual Robot so that after giving the shuttle it can start its automation.
- Two Baumers were on the front side of Automatic bot mounted on edges of left and right sections used to detect the borders of the field while aligning on the throwing positions.

Issues:

- Due to slight oscillations in shuttlecocks about its pivot point on the Automatic bot it was sometimes difficult to detect the shuttlecocks.
- Some calibration issues were faced while detecting the borders of the field.
- The presence of manual bot was not as a whole. It was being detected partially.

Solution:

- The problem for shuttlecock was solved by giving a small delay to Automatic bot and then shoot the shuttlecock whether or not the shuttle was detected by the Baumer.
- A white sheet was placed on Manual bot so that Baumer can detect it as a whole object.



Modifications:

- Initially Baumers and Sharp sensors were used for the alignment of Automatic by detecting the edges of the field. Later only on throwing positions Baumers were used for detecting the edge of the field as mentioned above in the third bullet of the uses.
- For alignment only Sharp sensors were used.

Sharp sensors:

Sharp GP2Y0A02

Description:

GP2Y0A02 is a distance measuring sensor unit, composed of an integrated combination of PSD (position sensitive detector), IRED (infrared emitting diode) and signal processing circuit. The variety of the reflectivity of the object, the environmental temperature and the operating duration are not influenced easily to the distance detection because of adopting the triangulation method. This device outputs the voltage corresponding to the detection distance. So this sensor can also be used as a proximity sensor

Features:

- Operating voltage: 4.5v to 5.5 volt
- Average current consumption: 33mA(Typical)
- Distance measuring range: 20cm to 150cm
- Analog output type
- Package size: 29.5×13×21.6 mm

Use:

- It was used on Automatic Robot(AR) for alignment of AR.
- Total 4 sensors were used there

Issues:

- Unstable values because of insufficient supply.
- No values in between.
- Sensors changed many times.

Solutions:

- In order to stabilize power supply lines, insert a by-pass capacitor of 100uF or more between VCC and GND.
- Use plastic mount while mounting on the bot.
- Avoid it from static charge; use double sided tape properly before mounting.
- The lens of this device needs to be kept clean. There are cases that dust, water or oil and so on deteriorate the characteristics of this device.
- In case of changing the mounting angle of this product, it may measure the distance exactly.

Others:

- Autonics Sensor:

Cytron LSA 08:

LSA08 need to be calibrated to retrieve the dark and bright value of the surface that it will do the line follow. Every of the sensor pairs need to be exposed to the dark and bright surface for LSA08 to read and save the value. The surfaces brightness value will be saved to non-volatile memory of LSA08. Hence, only onetime calibration is needed for the same background and line unless the background and the line changed, then recalibration is needed. This was not as reliable as image processing. Hence camera was used for line detection.

PCB Manufacturing Process

Manufacturing PCB is an essential task for any of the electronics circuit the circuit board are permanently mounted on PCB by soldering which provides them a long and durable life

The following are the steps for PCB MANUFACTURING and development

- Clad Shearing:

  Clad Shearing or cutting can be done using hex saw or by any type of Shearing machine or simply just by paper cutter

- Clad cleaning:

  Clad cleaning needs a b grade cleaning scrubber and detergent it can also be done using the sharp edge of paper cutter.

- Clad printing:

  The artwork can be printed on the clad.

With help of press or laminator and cool it down with water.

- Etching:

  After the pattern transfer is done the clad is ready to etching in chemical of $FeCl_3$ or ammonia

- Stripping:

  The etched clad now is to be stripped and the non-conducting coatings can be removed from

PCB.

- Drilling:

  The PCB can be drilled and can be made ready for component mounting.

- Mounting:

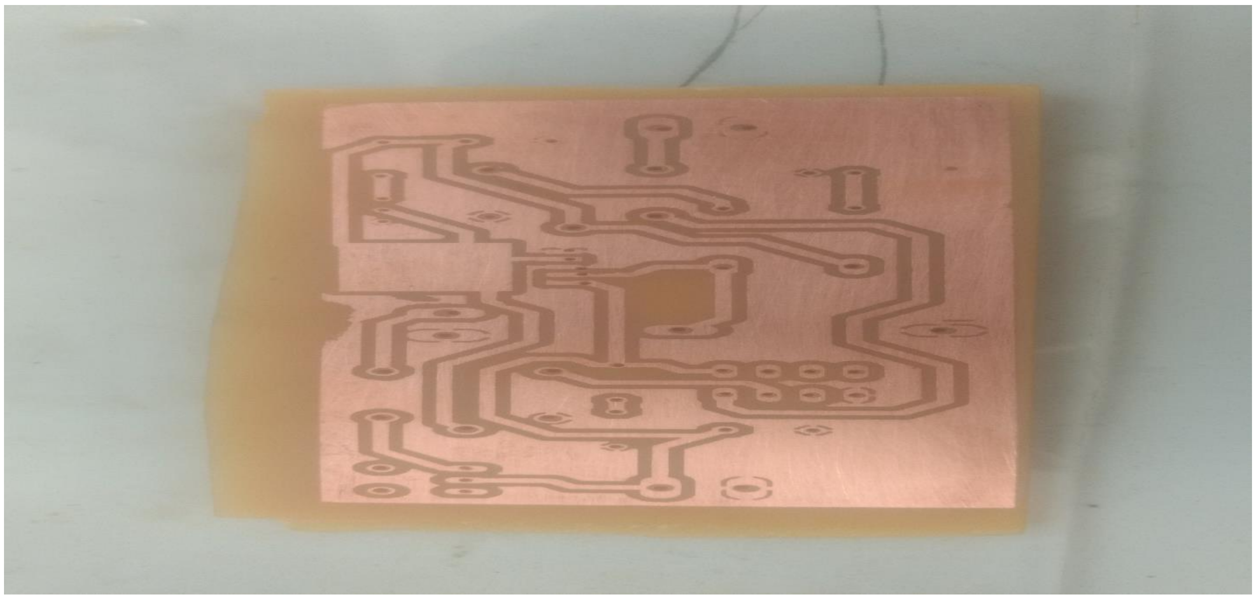
  The is placing of component on its place with appropriate polarity and place.

- Soldering:

  The last step is to solder your device with soldering iron and smd soldering with hot air gun.

- Solder mask:

The optional step to provide your circuit extra protection with green solder mask with help of UV curable solder masking ink



An example of a full PCB

Tools used in PCB fabrication

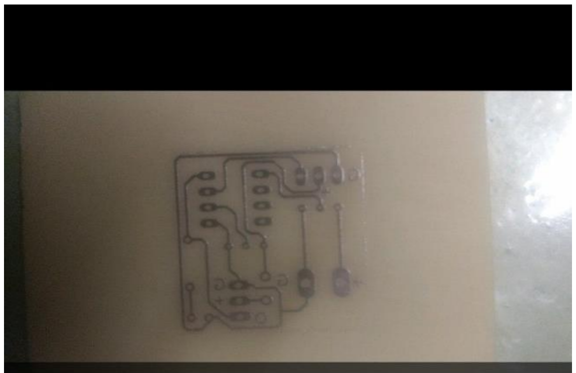
- FeCl3
- Drill machine
- 0.6mm,0.8mm,1mm,2mm drill bit
- Heater
- Glass epoxy copper clad
- Paper phenolic copper clad

The PCB manufactured in lab are

1. Buck Boards
2. Breaking board
3. Distribution board
4. Pneumatic actuation board
5. AR shield
6. MR shield
7. Sabertooth protection circuit

Debugging issue:

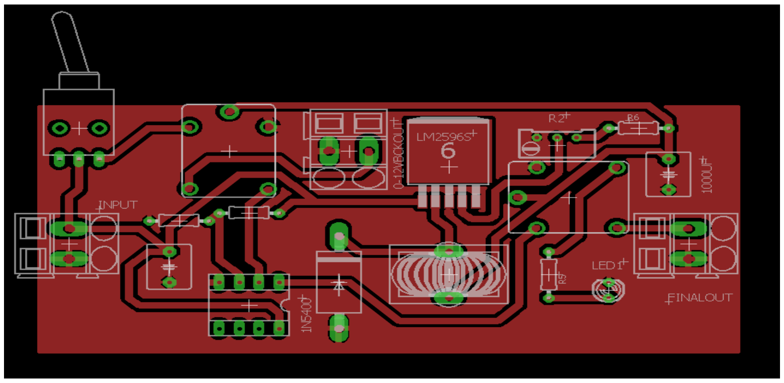
- Mismatch of solder combination
- Fabrication of thin tracks
- Drilling of fine holes
- Transferring of pattern on copper clad
- Tinning of whole board
- SMD soldering


Etched pcb tachometer

Power sources:

We used Orange 5000Mah and 3000Mah batteries for Sabertooth power supply and all other circuits power supply respectively. To supply 5V,10V,12V we used buck converter switched mode power supply which gives ripple free dc output voltages. We personally designed buck regulator for these applications.

It also has overvoltage and under voltage protection.



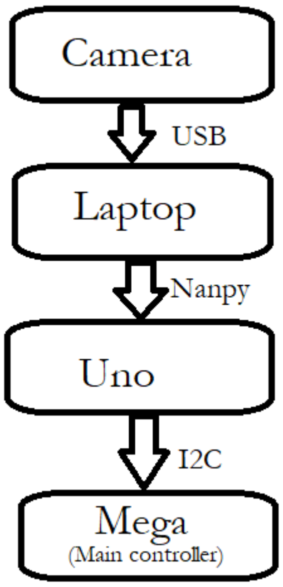
COMPUTER

Automatic Robot

Introduction:

This year's theme which was given by Vietnam had more focus on automatic robot. Major task was to be done by Automatic Robot. Automatic robot had to go to throwing zones and shoot the shuttlecock. For which we had to trace a line to reach our destination. But instead of choosing LSA08 line tracer from Cytron Technologies, we opted to use a Camera (PS3 Eye) and use OpenCV to detect the line. We are the only team who had used Image Processing for Automation.

Flowchart:



Camera was connected to laptop. Laptop did all the heavy task of Image Processing and Data extraction. Once we get the info about line and position of robot then these values were passed to Nanpy whose task was to send those values from python code to Arduino Uno attach to that Laptop. Then this Uno was connected to our main Controller board which was an Arduino Mega with I2C.

Image Processing

Camera Input

The camera we used is PS3 Eye since it provides a frame rate as high as 150 fps, which made it ideal for our use in the AR bot for the purpose of line tracing.

The source code for extracting and processing the feed obtained from the camera is written in Python.
The source code is primarily divided into three files in order to make it modular, hence improving its readability, and making it easier to maintain and debug it.

The three files are: -

Credits.py

This file contains all the constants that we will be dealing with in the entire source code such as the resolution of the camera input. Hence it is aptly named 'Credits'.

Master.py

This file contains the source code where relevant data is extracted and processed from the camera and sent to the Arduino Mega.

Libraries used: -

Initially, we make use of the video4linux application's command line tools to disable the automatic white balance and auto exposure settings, and to set our own suitable parameters manually.
This was necessary as it was observed that change in exposure interfered with our filters and generated noise.

The OpenCV built-in functions that we've used are as follows: -

cv2.videoCapture(cameraIndex)

Returns a video capture object.

VideoCaptureObject.set()

Allows us to set, among other things, the resolution and frame rate of the video stream.

VideoCaptureObject.read()

Returns a Boolean variable along with an individual frame of the video stream.

cv2.cvtcolor()

Allows us to convert frame which is initially in BGR format into the HSV format. This is because the line that we wish to trace is white in color. And HSV format makes it easier to extract any single color from the screen.

cv2.inRange()

Used to define the range of the mask that will be applied later on the frame.

cv2.bitwise_and()

Used to apply the mask on the frame, thereby extracting a custom range of white from the frame.

cv2.morphologyEx()

This function is used to apply the two morphological transformations of opening and closing.
Opening helps in removing any residual noise outside the principal object i.e. the line in our case.
Closing helps in plugging any holes inside the line.

cv2.imshow(windowname,image_name)

Used to display the image on the screen. This is useful for checking if any form of noise is still present in our image. It proves extremely helpful in debugging and also calibration.

cv2.waitkey(timeinterval)

Displays the image for specified number of seconds

cv2.destroyAllWindows()

Closes all and any image windows.

Numpy built-in functions used: -

np.array()

Used to define the lower and upper range of white that is to be extracted from the frame.

np.ones()

Used to set the kernel size which is needed to perform the morphological operations of opening and closing.

User defined functions used: -

fpsCount()

Gives us the number of frames generated in one second. Used to monitor the effect of applying filters to the frames on the frame rate. The frame rate on the primary laptop was observed to be 35-45 fps while the same for the backup laptop was 40-50 fps

Camera.py

Contains functions which are used to determine the position of line and also to detect a plus.

The functions are as follows: -

scan_up(image_name)

Scans all the columns on a row at the top twice; once from the left and once from the right for maximum intensity and returns the average of the two values. This gives us the position of the line on the top.

scan_down(image_name)

Scans all the columns on a row at the bottom twice; once from the left and once from the right for maximum intensity and returns the average of the two values. This gives us the position of the line on the bottom.

scan_left(image_name)

Scans all the rows on a column on the left twice; once from the top and once from the bottom for maximum intensity and returns the average of the two values. This gives us the position of the line on the left.

scan_right(image_name)

Scans all the rows on a column on the right twice; once from the top and once from the bottom for maximum intensity and returns the average of the two values. This gives us the position of the line on the right.

plus(thresh_name)

Used to check for a cross, which helps in recording the position of the bot.
If all the above four functions return non-zero values, it returns 1 indicating that a cross has been detected or else it returns 0.

Nanpy

Image Processing was primarily used for Line Tracing, Cross Detection and Aligning.
Four Scan Lines Viz. Top, Down, Left, Right which represent various positions of the view did this task effectively. Each scan Line returned a value. Mapping the values from the four scanlines ultimately laid us to the final position of the Line. The nature of value returned is mentioned in the Python Section.

Ideally the values would return perpetually. But even if the values are not returned for an instance, the system should handle the condition. Thus Master-Slave model was used for the Automatic Robot. Arduino Uno acted as a Slave, who sent the values to the Master using Nanpy. The Arduino Mega acted as Master who ran all the functions using the values it got from the Slave Uno.

Previous Attempts: -

To accomplish the above task, we tried writing custom code.
A 12-digit value which denoted 3 digits chronologically to up, down, left, right scan line respectively.

This code gets the values available on the serial and send it whenever requested. If the request comes before all values are acquired in the variable a dummy value of 999999999999 is sent, which is later ignored by the receiver.

```
#include <Wire.h>
char ch[12]="999999999999", val[12],count=0;
void setup() {
 Wire.begin(8);              // join i2c bus with address #8
 Wire.onRequest(requestEvent); // register event
 Serial.begin(9600);
}

void loop() {
 if(Serial.available())
 {
  count = 0;
 while(Serial.availableForWrite())
 {
  val[count]= Serial.read();
  count++;
  }

  Serial.println(val);
  }
  delay(10);
}

void requestEvent() {
  if(count==12)
    Wire.write(val);
  else
    Wire.write(ch);
}
```

Reciever Side:

```
#include <Wire.h>

void setup()
{
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(115200);  // start serial for output
}
char ch[3],cu[3],cr[3],cd[3],cl[3];
int up,right,down,left;
String u,r,d,l;
void loop()
{
  Wire.requestFrom(8, 12);    // request 6 bytes from slave device #2

  while(Wire.available())    // slave may send less than requested
  {
    for(int i=0;i<3;i++)
    {
        char c = Wire.read();
        ch[i] = c;
        u = ch;
    }
    for(int i=0;i<3;i++)
    {
      char c = Wire.read();
      ch[i] = c;
      r = ch;
    }
    for(int i=0;i<3;i++)
    {
      char c = Wire.read();
      ch[i] = c;
      d = ch;
    }
    for(int i=0;i<3;i++)
    {
      char c = Wire.read();
      ch[i] = c;
      l = ch;
    }
    up = u.toInt(); right = r.toInt(); down = d.toInt(); left = l.toInt();

    if(up!=999 && down!=999 && right!=999 && left!=999)
    {
     Serial.print(up);
     Serial.print("-");Serial.print(right);
     Serial.print("-");Serial.print(down);
     Serial.print("-");Serial.println(left);
     delay(100);
    }
    //else
     //Serial.println("ignored");
  }
}
```

This code successfully segregates the 12-digit value into corresponding classes.

Problems:

Though the code worked fine, we faced issues of hanging. The reason for the hanging may be buffer overload, Serial overload.
The solution to this problem was not established.
Therefore, it was decided to use a well-established already written library NANPY.

Rewriting Nanpy Library:

Nanpy is a library that use your Arduino as a slave, controlled by a master device where you run your scripts, such as a PC, a Raspberry Pi etc.

The main purpose of Nanpy is making programmers' life easier, providing them a powerful library to create prototypes faster. Nanpy is easily extensible and can theoretically use every library, allowing you to create how many objects you want.

The Image Processing framework used OpenCV Library with Python interface. The objective was to get the values on Serial. Then the slave would acquire them and sent them to the master. As Nanpy works on the same principle, we were successful getting the desired output with few modifications in the library.

Modifications: -
Let's have a look at Arduino's Wire slave sender example:

```
#include <Wire.h>

void setup() {
  Wire.begin(8);                // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
  Wire.write("hello "); // respond with message of 6 bytes
  // as expected by master
}
```

- So we need a requestEvent function that get called whenever the master request.
- We need extract top, down, left, right from the Wire.

The following modification is Made in the ArduinoClass.cpp file.

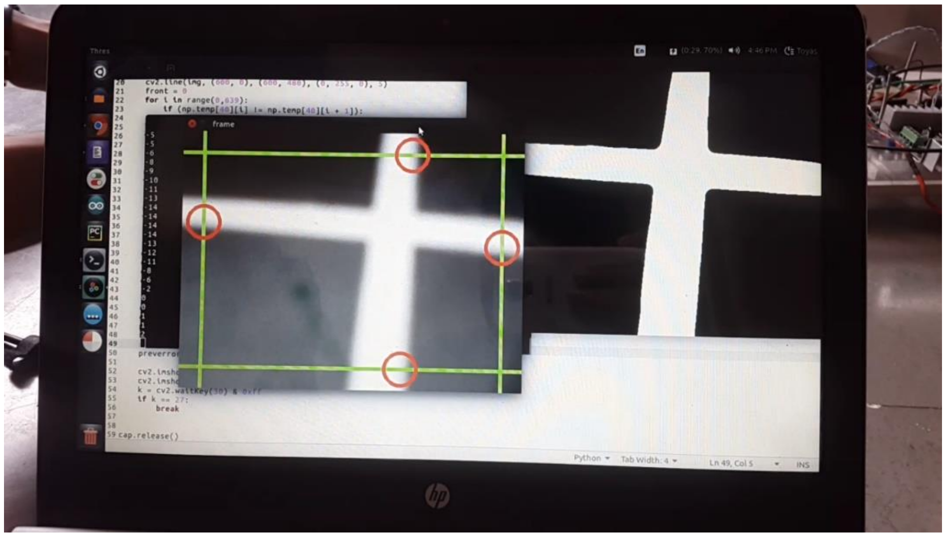
Modifications are in highlighted.

```
#include <Arduino.h>
#include "ArduinoClass.h"
#include <stdlib.h>
#include<Wire.h>
int up = 0, down = 0, left = 0, right = 0;
const char* nanpy::ArduinoClass::get_firmware_id()
{
  return "A";
}

void requestEvent() {

Wire.write(up>>8);
Wire.write(up);
Wire.write(down>>8);
Wire.write(down);
Wire.write(right>>8);
Wire.write(right);
Wire.write(left>>8);
Wire.write(left);

// respond with message of 6 bytes
  // as expected by master
}

void nanpy::ArduinoClass::elaborate( nanpy::MethodDescriptor* m ) {
  if (strcmp(m->getName(), "dw") == 0) { // digitalWrite
    digitalWrite(m->getInt(0), m->getInt(1));
    m->returns(0);
  }

  if (strcmp(m->getName(), "r") == 0) {  // digitalRead
    m->returns(digitalRead(m->getInt(0)));
  }

  if (strcmp(m->getName(), "aw") == 0) { // analogWrite
    int temp = m->getInt(0);
```

```
   if (temp < 20)
   {
    analogWrite(temp, m->getInt(1));
    m->returns(0);
   }
   else if (temp == 20)
   {
    Wire.begin(8);                // join i2c bus with address #8
    Wire.onRequest(requestEvent);
    m->returns(0);
   }
   else if (temp==21)
   {
    up = m->getInt(1);
    m->returns(0);
   }
    else if (temp==22)
   {
    down = m->getInt(1);
    m->returns(0);
   }
    else if (temp==23)
   {
    right = m->getInt(1);
    m->returns(0);
   }
    else if (temp==24)
   {
    left = m->getInt(1);
    m->returns(0);
   }
  }

  if (strcmp(m->getName(), "a") == 0) {  // analogRead
   m->returns(analogRead(m->getInt(0)));
  }

  if (strcmp(m->getName(), "pm") == 0) {  // pinMode
   pinMode(m->getInt(0), m->getInt(1));
   m->returns(0);
  }
  if (strcmp(m->getName(), "delay") == 0) {
    m->returns(0);
  }

  if (strcmp(m->getName(), "m") == 0) {  // millis
    m->returns(millis());
  }

  if (strcmp(m->getName(), "pi") == 0) {  // pulseIn
    pulseIn(m->getInt(0), m->getInt(1));
    m->returns(0);
  }

  if (strcmp(m->getName(), "s") == 0) {  // shiftOut
    // shiftOut(dataPin, clockPin, bitOrder, value)
    shiftOut(m->getInt(0), m->getInt(1), m->getInt(2), m->getInt(3));
    m->returns(0);
  }
};
```

Corresponding Python code:

```
    a.analogWrite(21, int(Camera.scan_up(thresh)))
    a.analogWrite(22, int(Camera.scan_down(thresh)))
    a.analogWrite(23, int(Camera.scan_right(thresh)))
    a.analogWrite(24, int(Camera.scan_left(thresh)))
```



```
    cv2.line(img, (600, 0), (600, 480), (0, 255, 0), 5)
    front = 0
    for i in range(0,639):
        if (np.temp[40][i] != np.temp[40][i + 1]):
```

```
-5
-5
-6
-8
-9
-10
-11
-13
-14
-14
-14
-14
-13
-12
-11
-8
-6
-2
0
0
1
1
2
```

```
    preverror
    
    cv2.imsh
    cv2.imsh
    k = cv2.waitKey(30) & 0xff
    if k == 27:
        break

cap.release()
```

Main Controller(Mega)

Now once we received values from I2C we need to use them in proper way.

Align Functions:

Align functions kept track of coordinate of points on line wheel-wise. Their purpose was to keep the wheel assigned to them on the line all the time. And then we call different maneuvering functions.

Eg.

```
void alignYCam()
{
 recvVal();
 if(camUp != -1)
 {
  int tempUp = yCamMid-camUp;
  int tempDown = yCamMid - camDown;
  tempUp *= Kpc;
  tempDown *= Kpc;
  if (camUp != 0)
   driveUp(-tempUp, true);
  else
  {
   if(upSide)
    driveUp(-speedLimit, false);
   else
    driveUp(speedLimit, false);
  }
  if (camDown != 0)
   driveDown(tempDown, true);
  else
  {
   if(downSide)
    driveDown(speedLimit, false);
   else
    driveDown(-speedLimit, false);
  }
  if(camUp != 0)
  {
   if (tempUp>0)
    upSide = true;
   else
```

```
      upSide = false;
    }
    if(camDown != 0)
    {
     if (tempDown>0)
       downSide = true;
     else
       downSide = false;
    }
  }
}
```

Similar function was written for X axis.

Maneuvering:

Maneuvering Functions were the highest level functions which held the main logic of automation. Nothing complex was done in those functions. Only underlying low level functions were called in a pattern. This pattern was decided on basis of current location of robot and desired location of it.

Eg.

```
void gotoTZ1()
{
  if(mode==2)
  {
    goLeft();
    giveMeABreak(350);
    goUp(1);
    giveMeABreak(350);
    correctUpnew();
    giveMeABreak(350);
    goRightnew(2);
  }
  else
    goRightnew(3);

  aglin();
   delay(2500);
  TZ1Piston();


  goDown(1);
```

```
 giveMeABreak(350);
 while(runTill(1500))
 {
   alignX1Cam();
 }
 giveMeABreak(350);
 goLeft(2);
 giveMeABreak(350);

 mode = 1;
}
```

The stop command was given to the robot as soon as the reference line required entered the field of view of the camera. Even as brakes were applied to the wheels, the momentum of the robot was so much that it did not stop at the required position. The inertia of the robot sometimes drove the reference line out of the field of view of the camera. This would cause a problem to the subsequent motion and commands as the reference line was either not present at all or not present at the position it was required at.

As an example, consider the following function. The purpose of the function is to correct the position of the robot after a “down” direction movement.

Pseudo code:

```
 function to correct after “downward” motion()
{
   get line values

   while (horizontal line is not completely visible)
   {
            get line values;

     if  (line under left wheel is not visible)
         move left wheel forward;
     else
         stop left wheel;

     if  (line under left wheel is not visible)
       move right wheel forward;
     else
```

```
      stop right wheel;
  }
}
```

Thus the robot is brought back onto the reference line with the help of this function to execute the following commands. If it is already on the reference line, no problem arises as the function is called, and the robot simply remains on the line without movie further "upward".
Similar functions were written for each of left, right and "upward" motion too.

The Inertia Problem:

At the throwing zone, a command to stop at the required plus was given. The inertia of the robot drove the robot a little beyond the required plus such that plus was out of the field of view of the robot. This meant that the correction function was called to get the robot back onto the required plus. As observed this consumed a time which could be decreased.

So, the robot was given the stop command at the plus before the throwing plus. The inertia of the robot drove it towards the throwing plus after the stopping command, and through a correction function it was moved further toward the throwing plus until it was visible to the camera.

This meant that speed of the robot while moving towards the throwing zones could be increased as the momentum and inertia was not hindering but, in turn, helping to stop at the required position. This further decreased time.

Pseudo code:

```
void goRightnew(parameter 'val' given)
{
  for (run 'val'-2 times)
  {
    get line values
    while (line under 'down' wheel is not visible)
    {
      align on X axis;
      move right with higher speed;
      get line values;
    }
    while (vertical line is not visible)
    {
      align on X axis;
      move right with higher speed;
      get line values;
    }
  }

  brake
    correction function();
  brake
}

correction function()
{
  get line values;
  while(vertical line is not visible)
    {
      get line values;
      if  (line under up wheel is not visible)
         move up wheel towards throwing plus;
      else
         stop up wheel;
      if  (line under down wheel is not visible)
         move down wheel towards throwing plus;
      else
         stop up wheel;
    }
  brake;
}
```

Manual Robot

Drive (X-Holonomic Drive)

A holonomic drive allows an operator to translate in any direction, independent of rotation. There are several ways to achieve this mechanically, such as through the use of Omni-wheels or mecanum wheels.

In driving a holonomic robot, it is convenient to have a function which takes a speed, direction, and rotation. This provides a clean interface to the drive for both autonomous programming and operator control. Such an algorithm takes the direction of travel and speed desired for the operation and calculates the speed to drive each wheel to achieve the desired vector. In addition, the algorithm takes in the desired rotation and adjusts the speed of each motor to achieve the given rotation while still maintaining the same velocity vector

Joystick Mixing:

An intuitive way to drive a holonomic robot is to have the robot drive in the direction that a joystick is pointed, with a magnitude based on the distance that the joystick is from neutral. If a twisting joystick is used, the amount of twist can control the rate of rotation of the robot. In the absence of a twistable joystick, the x-axis of a second joystick can be used to control the rate of rotation. This potentially allows one joystick control of the entire drive system, which reduces the number of input controls the driver must consider while directing the motion of the robot, allowing that much more attention to be on the actual path the driver wishes the robot to take.

Implementation:

To convert the x and y axis inputs of a joystick to magnitude and direction form, it is convenient to use formulas derived to convert Cartesian coordinates to Polar coordinates - namely r2 = x2 + y2 and theta = tan-1(y/x). This will return a radius (the magnitude), and an angle (the direction). The formulas are easily applied because the input from the joystick is in ranges from negative one to one, which is the same range as a unit circle, and therefore requires no conversion to use with trigonometric identities, particularly the tangent identity of opposite over adjacent, which would scale any values outside the x2 + y2 = 1 region (namely in the areas where the joystick goes to the corner regions where both the x and y inputs are near 1 or -1). To obtain the magnitude, the x and y inputs are squared and summed, and the square root of that quantity is the vector's magnitude. The direction can be calculated using the atan2 function from math.h. This function takes an x and y coordinate and returns the angle to that point in radians and does not break down around Pi/2 and -Pi/2, where a tradition tangent function approaches infinity due to the denominator (cos theta) approaching 0. This would be available for retrieval in degrees or radians, since the mecanum code takes its input in degrees, while the standard output of the atan2 function is in radians.

Driver Oriented Control:

A further intuitive system would be to use a gyro to track the current heading relative to the original heading, and to use this data to adjust the joystick input. The input would be adjusted to drive the robot in a direction relative to the driver, instead of relative to the robot itself. This means the driver would not have to adjust their commands to the robot mentally before sending them to the robot - when the driver move the joystick away from them, the robot always move away from them, no matter which way it is facing. Combined with joystick mixing, this control system is extremely intuitive, and requires less "housekeeping" consideration during a match.

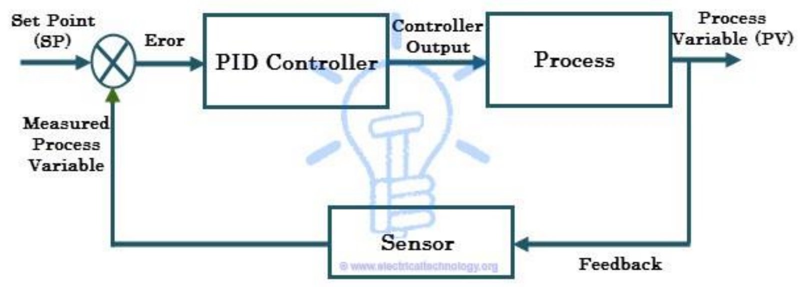
Implementation:

To implement driver-oriented control, the difference of the joystick's direction input and the heading returned by the gyro can be passed in to the HolonomicDrive() function, instead of just the joystick's calculated direction input. This combination adjusts the input for the current front of the robot, so the input and reaction of the robot is always relative to the driver's frame of reference.

PID

A combination of proportional, integral and derivative actions is more commonly referred as PID action and hence the name, PID (Proportional-Integral-Derivative) controller. These three basic coefficients are varied in each PID controller for specific application in order to get optimal response.

It gets the input parameter from the sensor which is referred as actual process variable. It also accepts the desired actuator output, which is referred as set variable, and then it calculates and combines the proportional, integral and derivative responses to compute the output for the actuator.



Consider the typical control system shown in above figure in which the process variable of a process has to be maintained at a particular level. Assume that the process variable is temperature (in centigrade). In order to measure the process variable (i.e., temperature), a sensor is used (let us say an RTD).

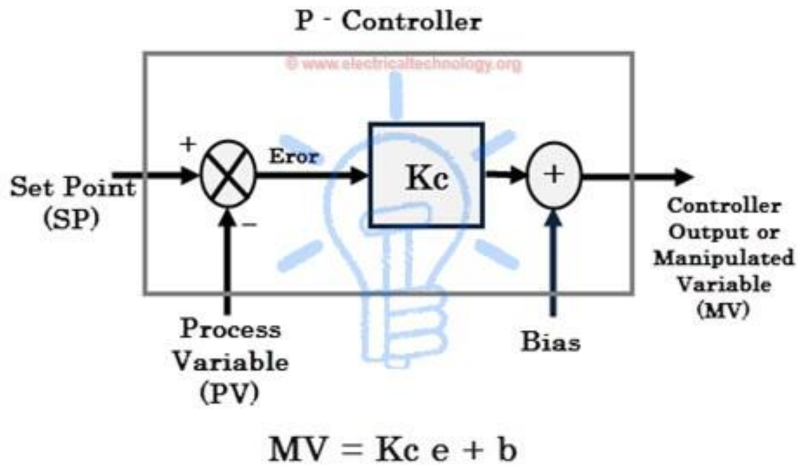
A set point is the desired response of the process. Suppose the process has to be maintained at 80 degrees centigrade, and then the set point is 80 degrees centigrade. Assume that the measured temperature from the sensor is 50 degrees centigrade, (which is nothing but a process variable) but the temperature set point is 80 degrees centigrade.

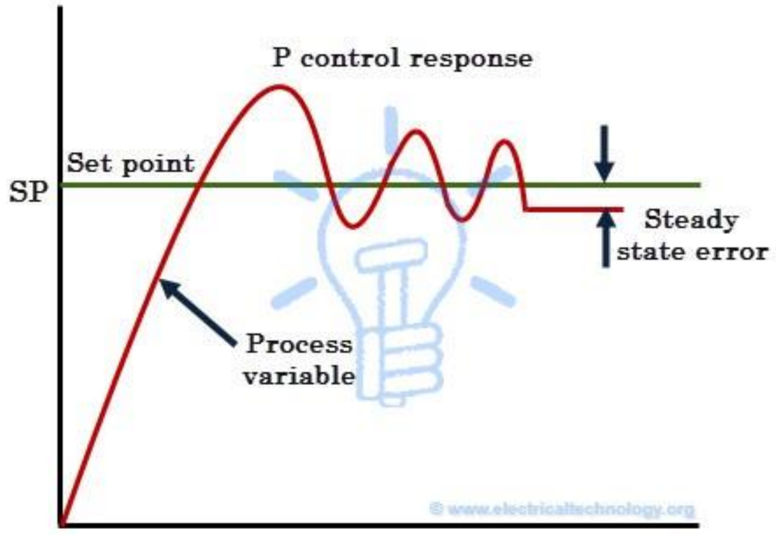
This deviation of actual value from the desired value in the PID control algorithm causes to produce the output to the actuator (here it is a heater) depending on the combination of proportional, integral and derivative responses. So, the PID controller continuously varies the output to the actuator till the process variable settle down to the set value. This is also called as closed loop feedback control system.

P-Control Response:

Proportional control or simply P-controller produces the control output proportional to the current error. Here the error is the difference between the set point and process variable (i.e., e = SP – PV). This error value multiplied by the proportional gain (Kc) determines the output response, or in other words proportional gain decides the ratio of proportional output response to error value.

For example, the magnitude of the error is 20 and Kc is 4 then proportional response will be 80. If the error value is zero, controller output or response will be zero. The speed of the response (transient response) is increased by increasing the value of proportional gain Kc. However, if Kc is increased beyond the normal range, process variable starts oscillating at a higher rate and it will cause instability of the system.

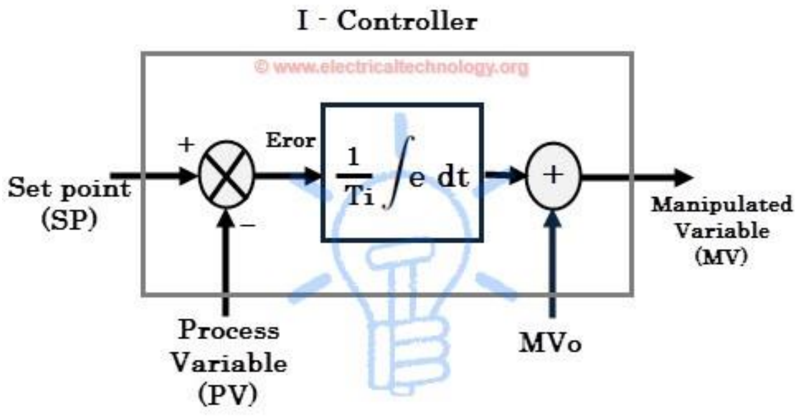

P - Controller
© www.electricaltechnology.org
Set Point (SP)
+
−
Eror
Kc
+
Process Variable (PV)
Bias
Controller Output or Manipulated Variable (MV)
MV = Kc e + b



Although P-controller provides stability of the process variable with good speed of response, there will always be an error between the set point and actual process variable. Most of the cases, this controller is provided with manual reset or biasing in order to reduce the error when used alone. However, zero error state cannot be achieved by this controller. Hence there will always be a steady state error in the p-controller response as shown in figure.

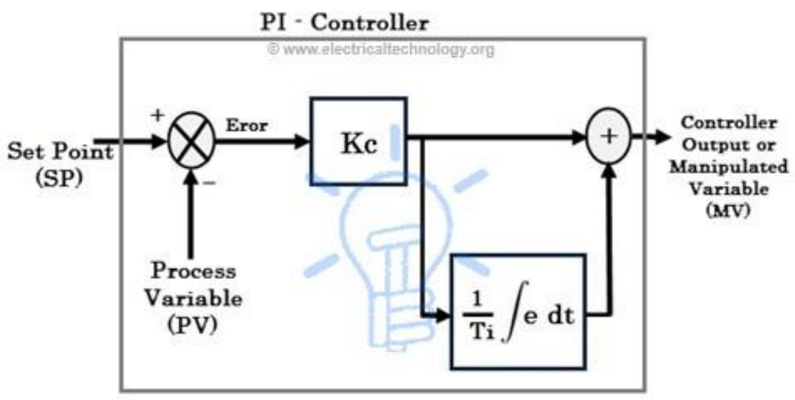
I-Control Response:

Integral controller or I-controller is mainly used to reduce the steady state error of the system. The integral component integrates the error term over a period of time until the error becomes zero. This results that even a small error value will cause to produce high integral response. At the zero-error condition, it holds the output to the final control device at its last value in order to maintain zero steady state error, but in case of P-controller, output is zero when the error is zero.



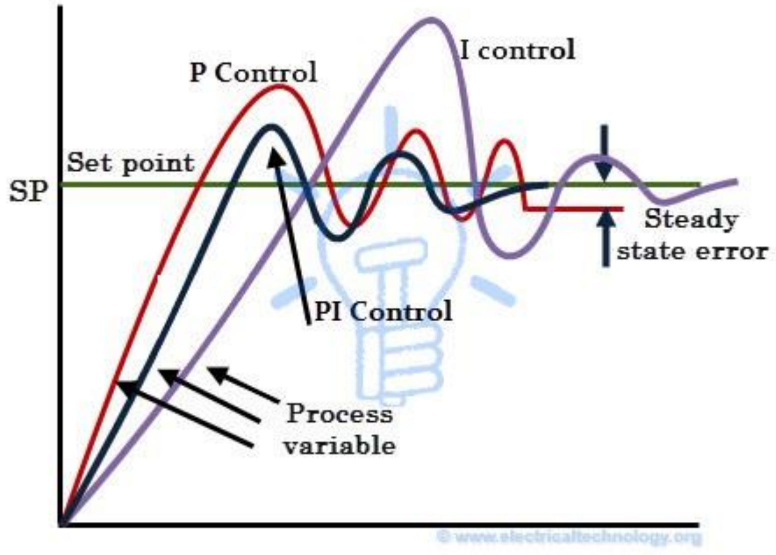
$$MV = \frac{1}{Ti}\int e\,dt + MVo$$

If the error is negative, the integral response or output will be decreased. The speed of response is slow (means respond slowly) when I-controller alone used but improves the steady state response. By decreasing the integral gain Ki, the speed of the response is increased.



$$MV = Kc\ e + \frac{1}{Ti}\int e\ dt$$

For many applications, proportional and integral controls are combined to achieve good speed of response (in case of P controller) and better steady state response (in case of I controller). Most often PI controllers are used in industrial operation in order to improve transient as well as steady state responses. The responses of only I-control, only p-control and PI control are shown in below figure.



D- Controller Response:

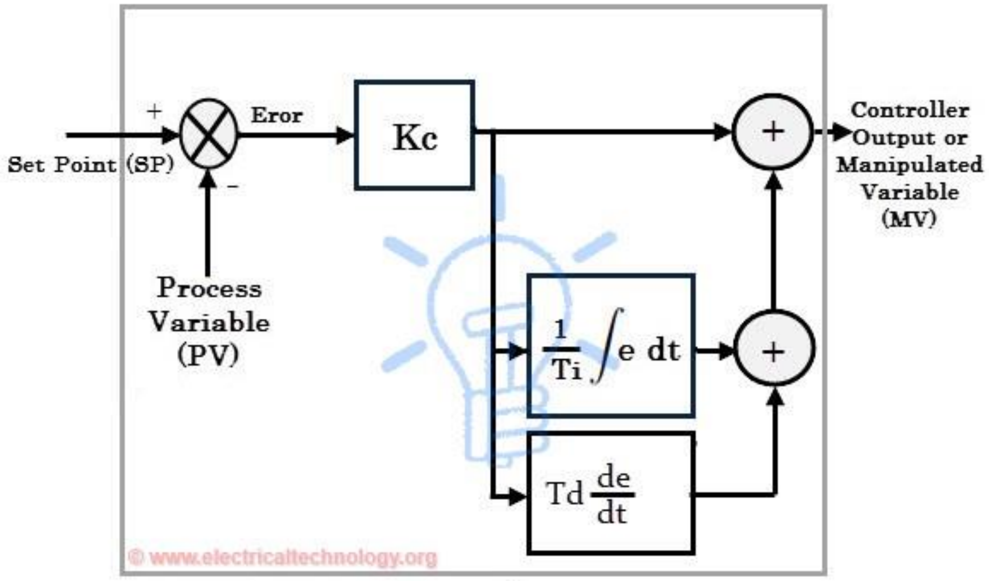
A derivative controller (or simply D-Controller) sees how fast process variable changes per unit of time and produce the output proportional to the rate of change. The derivative output is equal to the rate of change of error multiplied by a derivative constant. The D-controller is used when the processor variable starts to change at a high rate of speed.

In such case, D-controller moves the final control device (such as control valves or motor) in such direction as to counteract the rapid change of a process variable. It is to be noted that D-controller alone cannot be used for any control applications.

$$\text{output} = T_{\text{d}} \frac{\text{d}e}{\text{d}t}$$

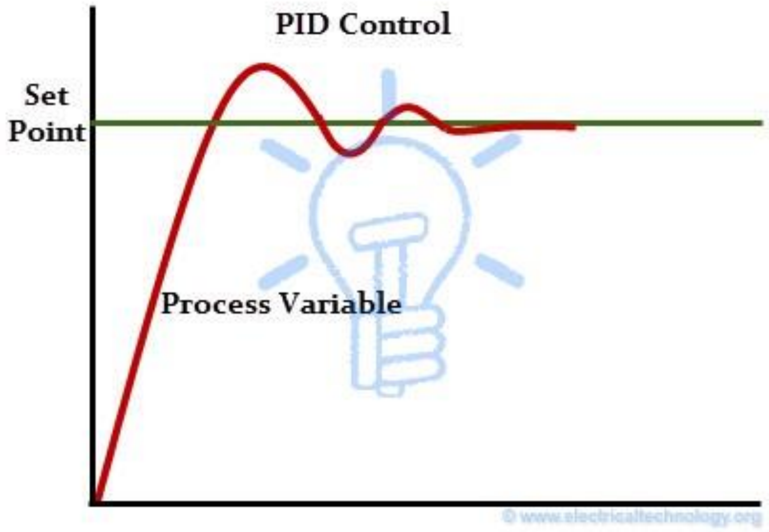
The derivative action increases the speed of the response because it gives a kick start for the output, thus anticipates the future behaviour of the error. The more rapidly D-controller responds to the changes in the process variable, if the derivative term is large (which is achieved by increasing the derivative constant or time Td).

In most of the PID controllers, D-control response depends only on process variable, rather than error. This avoids spikes in the output (or sudden increase of output) in case of sudden set point change by the operator. And also, most control systems use less derivative time td, as the derivative response is very sensitive to the noise in the process variable which leads to produce extremely high output even for a small amount of noise.



$$MV = Kc\ e + \frac{1}{Ti}\int e\ dt + Td\frac{de}{dt}$$

Therefore, by combining proportional, integral, and derivative control responses, a PID controller is formed. A PID controller finds universal application; however, one must know the PID settings and tune it properly to produce the desired output. Tuning means the process of getting an ideal response from the PID controller by setting optimal gains of proportional, integral and derivative parameters.



There are different methods of tuning the PID controller so as to get desired response. Some of these methods include trial and error, process reaction curve technique and Zeigler-Nichols method. Most popularly Zeigler-Nichols and trial and error methods are used.

This is about the PID controller and its working. Due to the simplicity of controller structure, PID controllers are applicable for a variety of processes. And also, it can be tuned for any process, even without knowing detailed mathematical model of process. Some of the applications include, PID controller-based motor speed control, temperature control, pressure control, flow control, level of the liquid, etc.

Shuttlecock

- Need and basic idea
    - According to the theme of Robocon 2018, this year the shuttlecocks which we are going to be throw through rings are made by teams themselves. So, basically we made them in lab by sewing.
    - There is need of two types of shuttlecock, normal and golden. There were 10 normal shuttlecocks and 5 golden shuttlecocks
    - Proper dimensions and shape

      Shape: spherical

      Diameter: greater than 12cm

      Weight: 60 to 100 grams

      String width: less than 1cm

      String length: greater than 25 cm
    - Materials

      Can be any natural or synthetic fibre
    - To get a proper trajectory so that it can pass through ring and perfectly fall in the bowl put in the opposite side of ring.
    - It should have 5 different colour fringes of 20 cm length measured from bottom of shuttle.
    - It should not bounce off the bowl.

- Trials

To make shuttlecock perfect according to the rules, we tried so many inside materials such that cotton, rice, beans, leaves, sand etc. Plenty of trials were taken to check those shuttlecocks. At the end, some parameters were fixed to make them.

- Design and manufacturing

First we were cutting and joining only 2 different parts but as it wasn't making a good spherical shape, we decided to use 6 different parts of almost eclipse shaped. Then join them using sewing machine. Strings attached to it was made up of three different bands. It was breaded, to increase its width and to get a proper trap. It has two notches one at 25 cm and other one at 35 cm from ball.

To make a normal shuttlecock we used 6 different clothes and for golden shuttlecocks, golden cloth was used.

- Normal shuttlecocks

  Shape: spherical

  Diameter: 12cm

  Weight: 87 grams

  String width: 1cm approx.

  String length: 35 cm

  Inside material: cotton

- Golden shuttlecocks

  Shape: spherical

  Diameter: 13 cm

  Weight: 96 grams

  String width: less than 1cm

  String length: 36 cm

  Inside material: 46 grams clay and remaining beans