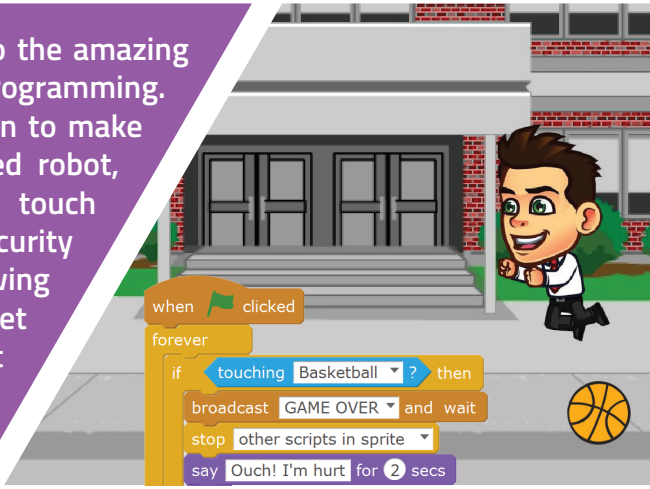
Starter Kit

DIY Learning Module

Starter Kit is a perfect start into the amazing world of robotics, electronics & programming. Unleash your creativity and learn to make your own smartphone controlled robot, interactive computer games, touch piano, pick and place robot, security system, alarm clock, light following robot, science experiments, pet feeder and much more. Lets get started and transform your ideas to reality.



learn better
build easier
debug smarter

evive

evive's Starter Kit | DIY Learning Module

Author:
Pankaj Kumar Verma
Co-Founder, Agilo Research Pvt. Ltd

Graphic Designer:
Dhrupal R Shah
Co-Founder, Agilo Research Pvt. Ltd

Creative Content Writer:
Shikhar Shivraj Jaiswal
Intern, Agilo Research Pvt. Ltd

Contact us:

B3 Rangdeep Apartment, Vijay Cross Roads
Navrangpura, Ahmedabad, Gujarat
India – 380009
Email: contact@evive.cc

www.evive.cc

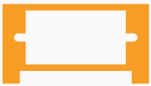
Please give your suggestions and feedback at contact@evive.cc

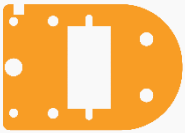
Printed in India

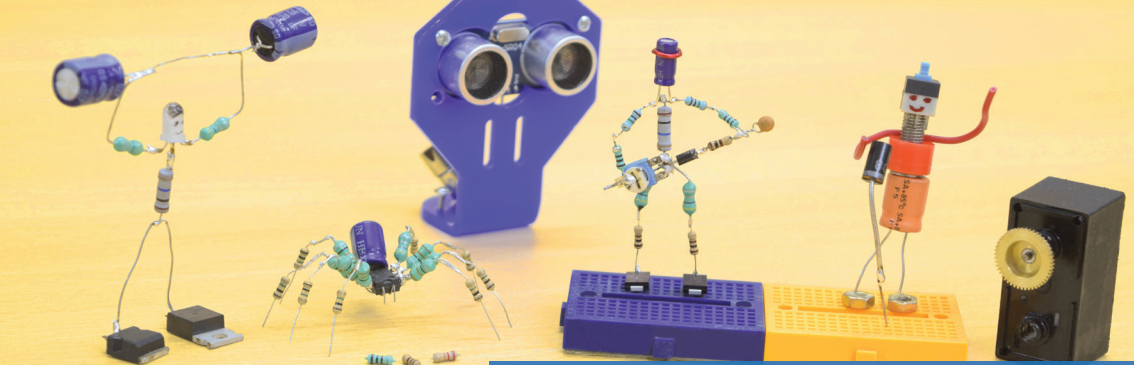
Contents

Component List

8 x Photo-resistor	
10 x Pushbutton	
LEDs 5 x Blue LED, 5 x Red LED 5 x Yellow LED, 5 x Green LED	
2 x Micro Servo with Accessories	
2 x 220 or 100 µF Capacitor	
Resistors 15 x 220 Ω Resistor, 10 x 1 kΩ Resistor 10 x 4.7 kΩ Resistor, 15 x 10 kΩ Resistor 10 x 1 MΩ Resistor	
6 x Alligator Clips	
1 x XY-axis joystick module KY-023	
1 x Bluetooth Module (HC05)	
Jumper Cables 20 x 20 cm Jumper (M-M) 20 x 20 cm Jumper (M-F) 20 x 20 cm Jumper (F-F)	
1 x Ultrasonic Sensor	

1 x Ultrasonic Holder	
1 x IR Sensor	
2 x Dual shaft BO Motors 6V, 50 RPM (with brackets)	
2 x Wheels	
1 x Base Plate	
1 x Base Servo Holder	
1 x Servo Mounting Plate	
1 x Servo Holder	
1 x Light Tracker Plate 1 1 x Light Tracker Plate 2	
1 x Gripper Claw 1	
1 x Gripper Claw 2	

1 x Gripper Base Plate	
1 x Gripper Plate 1	
1 x Gripper Plate 2	
1 x Gripper Link	
Fasteners 6 x M2 bolts of 12mm Length 10 x M3 bolts of 12mm Length 6 x M3 bolts of 25mm Length 25 x M3 bolts of 8 mm Length 4 x M4 bolts of 16mm Length 6 x M2 Nut 25 x M3 Nuts 4 x M4 Lock Nuts	
Standoffs 2 x 15mm M3 (Male to Female) 5 x 20mm M3 (Female to Female) 4 x 30mm M3 (Male to Female)	
Cable Ties	
1 x Castor Wheel	
Book	
1 x Screw Driver Star-head	
10 x Ice Cream Stick	



CHAPTER 2

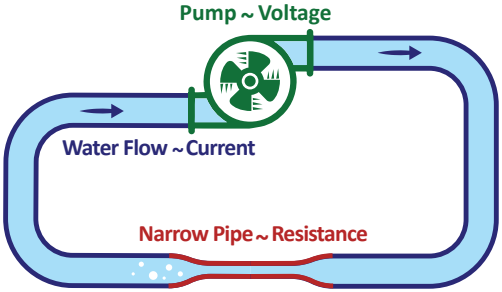
Basics of Electronics

Electricity, in short is defined as the flow of electric charge. However, this statement hardly gives any insight and leaves many questions unanswered – How does one cause charge to move? Where does it move? And so on. We will be dealing with current electricity throughout the course of this book so it is important to understand the basics of charge flow and all important terms associated with it.

Some Basic Concepts

Voltage, current and resistance are the three basic building blocks of electronics and electricity. In brief they can be defined as:

- VOLTAGE (V) – The potential difference between two points. Voltage is the driving force that pushes electricity to flow.
- CURRENT (I) – It is the rate at which charge is flows. If more number of electrons flow through a given point, then more current is said to be flowing.
- RESISTANCE (R) – It is defined as a material's tendency to resist the flow of charge (current). It is basically a measure of the difficulty to pass an electric current through that conductor.



At first, these concepts are difficult to comprehend because we cannot "see", i.e. physically observe them. We cannot see through the naked eye, the current flowing through a wire or how the wire through which it is flowing, is "resisting" its' flow. However, these can be explained through very simple analogies:

Consider water flowing though a pipe with variable thickness. The water inside the pipe is circulated using a water

pump. The water pump creates pressure which forces water to flow inside the pipe. The pressure at the across the pump represent voltage. The water inside the pipe represents charge. The more water pressure across the pump, the higher is the charge flow.

Battery and Voltage

The water pump is like a battery. For simplicity, let us have a battery connected to the pump. If the pump runs for more time the battery drains out and the pressure difference across the pump decreases (or voltage decreases). We can think of this decreasing voltage as, like when a flashlight gets dimmer as the batteries run down. There is also a decrease in the amount of water that will flow through the pipe. Less pressure means less water is flowing, which brings us to current.

Current

The amount of water flowing through the pipe is equivalent to current in electronics. The higher the pressure, the more the water flow, and vice-versa. In this particular model, we could measure the volume of the water flowing through pipe over a certain period of time. While dealing in current electricity, we measure the amount of charge flowing through the circuit over a period of time. Current is measured in Amperes. Current is represented in equations by the letter “I”.

Let us consider two identical loops with identical pumps which have same amount of water pressure, but have different pipe diameter. They both will have same amount of pressure the pump, but when the water starts flowing, the flow rate of the water in the narrower pipe will be less than the flow rate of the water in the wider pipe. In electrical terms, we say the current through the narrower pipe is less than the current through the wider pipe.

If we want the flow to be the same through both pipes, we have to increase the pressure across the pump with the narrower pipe. This increases the pressure (voltage) across the pump, pushing more water (current) through the pipe. Therefore we see that an increase in the voltage across the terminals of a wire increases the current flowing through it.

Resistance

A narrow pipe ‘resists’ the flow of water through it (because if a pipe is narrow, naturally less water will flow through it) even though the water is at the same pressure in the wider pipe. This is analogous to resistance. Its unit is Ohm and is represented by ‘R’.

In electrical terms, this is represented by two circuits with equal voltages and different resistances. The circuit with the higher resistance will allow less charge to flow, meaning the circuit with higher resistance has less current flowing through it.

Ohm’s Law

Combining the elements of voltage, current, and resistance, Georg Ohm developed the formula:

$$V = IR$$

An easy way to remember the formula for Ohm's Law is to think about a Vulture, rabbit, and an Indian. The rabbit will see the vulture flying over the Indian hence R=V/I. The Indian sees the vulture flying over the rabbit so I=V/R. The vulture sees the Indian and the rabbit next to each other so V=IR.

Continuing our discussion, the relations between 1 volt, 1 amp and 1 ohm, can be written as:

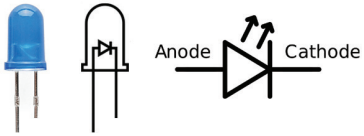
1 Volts = 1 Amp * 1 Ohm

If we go back to our analogy, let us defined the water pressure (voltage) is 1 volt, and the resistance is 2 ohms, then using ohms law we can get the current flowing:

I = V/R

I = 0.5 A

So, the current is lower in the pipe with higher resistance.



Light-Emitting Diodes (LEDs)

LEDs are like tiny light-bulbs. However, LEDs require a lot less power to light up by comparison.

LED User Guide

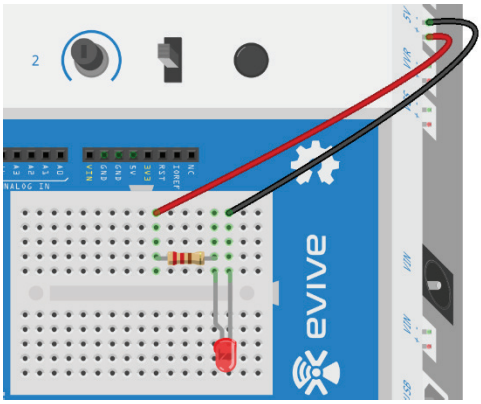
- **Polarity Matters:** In electronics, polarity indicates whether a circuit component is symmetric or not. LEDs only allows current to flow in one direction. And when there's no current-flow, there's no light. The positive side of the LED is called the "anode" and is marked by having a longer "lead," or leg. The other, negative side of the LED is called the "cathode." Current flows from the anode to the cathode and never the opposite direction. A reversed LED can keep an entire circuit from operating properly by blocking current flow.
- **More current equals more brightness:** The brightness of an LED is directly dependent on how much current it draws. That means two things. The first being that super bright LEDs drain batteries more quickly, because the extra brightness comes from the extra power being used. The second is that you can control the brightness of an LED by controlling the amount of current through it.
- **Current limiting:** If you connect an LED directly to a current source it will try to dissipate as much power as it's allowed to draw, and it will destroy itself. That's why it's important to limit the amount of current flowing across the LED. Resistors limit the flow of electrons in the circuit and protect the LED from trying to draw too much current.

Project: Glow a LED

Using the above theory, we will make a basic circuit, where we will glow a LED.

LEDs are fragile and can only have a certain amount of current flowing through them before they burn out. They have a current rating which is the maximum amount of

current that can flow through the particular LED before it burns out. To limit the current we will use resistors in the circuit to limit the current. The circuit given below shows how to power an LED.



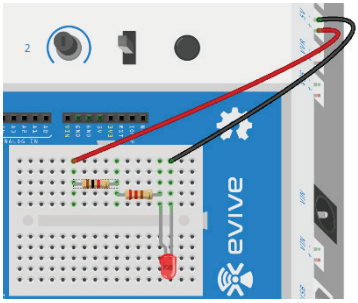
Generally the current rating of an LED is 20mA. Considering that the voltage drop across the LED is almost half the voltage provided to the circuit which is 5V, we can calculate the resistance value required to glow the LED. For this current the resistance value will be:

$$R = (V - V_L)/I$$

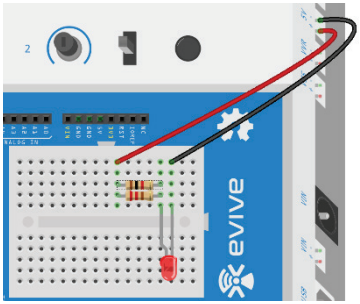
Now, V is 5V, V_L is 2.5V and I is 20mA, which makes R equal to 125 Ω. Hence, we always have to use a resistor greater than 125 Ω to safely run the LED.

We have 220 Ω resistor in our kit, hence we will use it to glow the LED.

The white colored dotted thing in above figure is a breadboard, which is used for easy wiring. All the vertical holes are connected internally, in two segments



Resistors in Series



Resistors in Parallel

Explore more

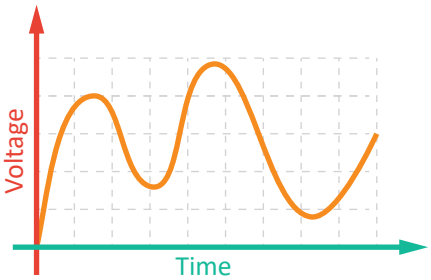
1. Change the resistance in the circuit and observe the change in the brightness of the LED.
2. Try **series** and **parallel** resistor combination (you can learn them online) and observe what happens when you have two resistor (220 Ω and 1 kΩ) in series and parallel configuration. The series and parallel circuit diagrams are shown above.

Hint: *In both the cases, try to find out which resistor will dominate more. Carefully note the brightness of LED in both cases.*

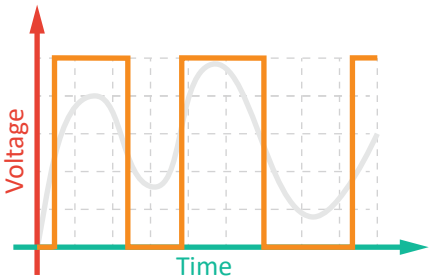
Analog and Digital Signals

A signal as referred to in electronics is a type of information carrier that "conveys data about the behavior or attributes of some phenomenon. Signals are time-varying. In electrical engineering the quantity that's time-varying is usually voltage. So when we talk about signals in this book, just think of them as a voltage that's changing over time.

Analog Signals



Analog Signal



Digital Signal

Its' easier to analyse a time-varying signal on a graph where time is plotted on the horizontal, x-axis, and voltage on the vertical, y-axis. Looking at a graph of a signal is usually the easiest way to identify if it's analog or digital; a time-versus-voltage graph of an analog signal should be smooth and continuous (the image on the left). While these signals may be limited to a range of maximum and minimum values, there are still an infinite number of possible values within that range.

Example: Video and audio transmissions are often transferred or recorded using analog signals. Pure audio signals are also analog. The signal that comes out of a microphone is full of analog frequencies and harmonics, which combine to make beautiful music. Sound is basically an analog signal that degrades over time and place.

Digital Signals

Digital signals must have a finite set of possible values. The number of values in the set can be anywhere between two or a very large number that's not infinity. Most commonly digital signals will be one of two values – like either 0V or 5V (the image on the right).

Logic levels

In digital electronics, there are only two states – ON (High or True) or OFF (Low or False). Using these two states, devices can encode, communicate, and control a great deal of data. Logic levels, in a general sense, describe any specific, discrete state that a signal can have. In digital electronics, we generally restrict our study to two logic states - Binary 1 and Binary 0. In simple words, signals can only have two values as far as digital electronics is concerned.

A logic level is a specific voltage or a state in which a signal can exist. We often refer

to the two states in a digital circuit to be ON or OFF. Represented in binary, an ON translates to a binary 1, and an OFF translates to a binary 0. Sometimes, we also call these states HIGH or LOW, respectively.

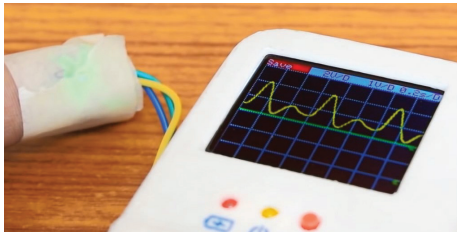
Most of the signals are analog. This conversion from analog to digital and vice versa is not simple and vary for input and output. The analog signal is broken into three ranges: LOW, HIGH and floating (where the signal can be HIGH or LOW).

These are the ranges an analog signal for input:

- 0V – 0.8V is LOW
- 2V – 5V is HIGH
- 0.8V – 2V is floating

For an output signal:

- 0V – 0.5V is LOW
- 2.4V – 5V is HIGH
- 0.5V – 2.4V is floating



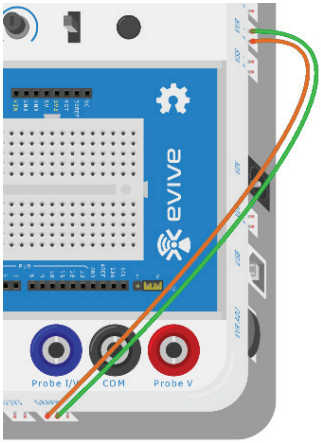
Analog Signal: Visualizing Heartbeat on evive Mini Oscilloscope

Project: Visualization of Analog and Digital Signals

In this project, we will visualize the analog and digital signal in evive Mini Oscilloscope. Before getting started make sure that you have evive firmware installed, otherwise you can download and upload it in your evive from http://evive.cc.

Visualizing Analog Signals

We are going to visualize how the variable voltage changes when we rotate the variable voltage knob located at the right face bottom (where Var Voltage is written). Connect the variable voltage supply to channel 2 of oscilloscope similar to the fritzing diagram given on left.



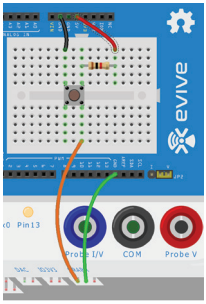
Turn on evive, go to Mini Oscilloscope and now you can see a yellow line which is the variable voltage output. Change the voltage by rotating knob and you can observe that the signal has infinite possible values.

Visualizing Digital Signals

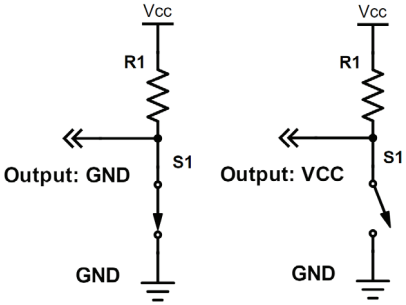
We will visualize the output of a tactile switch on the mini oscilloscope.

Switch is a device used to control the flow of current in a circuit. Switches are essentially binary devices: they are either completely ON (closed) or completely OFF (open). Therefore, it controls the open-ness or closed-ness of an electric circuit. When a switch is open, the circuit is broken and no current flows though the circuit, but when the switch is closed current flows through it.

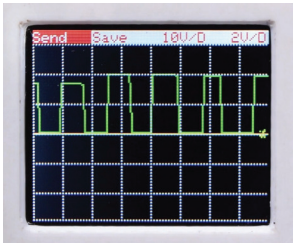
VCC is a non-zero higher potential (+5V) and ground is at zero potential.



Circuit Diagram



Pull up Resistors



Digital Signal visualized on evive Mini Oscilloscope

While using switches we measure the output from the switch. If there is nothing connected to the pin and you read the state of the pin, will it be high (pulled to VCC) or low (pulled to ground)? It is difficult to tell. This phenomenon is referred to as floating. To avoid this unknown state, a pull-up or a pull-down resistor ensures that the pin is in either a high or low state, while also using a low amount of current. While using switches you generally use pull-up or pull-down resistors.

With a pull-up resistor, the output will read a high state when the button is not pressed (since right now, it is connected to VCC). In other words, a small amount of current is flowing between VCC and the output pin (not to ground), thus the output pin reads close to VCC. When the button is pressed, it connects the input pin directly to ground. The current flows through the resistor to ground (remember, current chooses the path of minimum resistance), thus the input pin reads a low state.

Hence when the switch is not pressed, you get a HIGH reading on the digital pin and when the switch is pressed you get a LOW reading, when you are using pull-up resistor. Generally you use a resistor of 10k ohm. If you are using a pull-down configuration, the result is exactly opposite.

For pull-up resistor – "switch not pressed = HIGH"; "switch pressed = LOW'. The opposite is true for a pull-down resistor.

Let's visualize it. Make the circuit on the top. If you press the switch, output will be 0V and if it is not pressed, output will be 5V. Now try to implement pull-down resistor.

Conclusion

In this chapter, you have learned about the basic concepts of electrical circuits: Voltage, Current and Resistance, through tank-water analogy. Then you learned about the relationship between voltage, current and resistance through Ohm's law. You implemented the learned concepts to find out the minimum resistance required to glow a LED safely without burning it out. In the end, you learned about the difference between analog and digital signals. You also did a small project where you visualized analog and digital signal on evive Mini Oscilloscope. This chapter was the foundation for the upcoming chapters and projects.

Chapter 5
Dodge

Introduction

The aim of the game is to control the kid and dodge the objects coming his way for as long as possible. You can't dodge them forever; because basically we are playing a game of survival and the challenge essentially is to stay in the game for as long as possible. Let's begin by explaining the rules of the game through some figures.

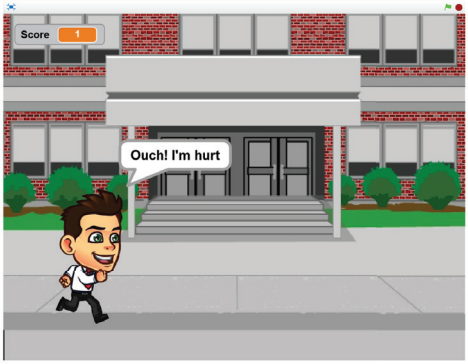
When there are no obstacles

The ball will come bouncing towards our "protagonist"

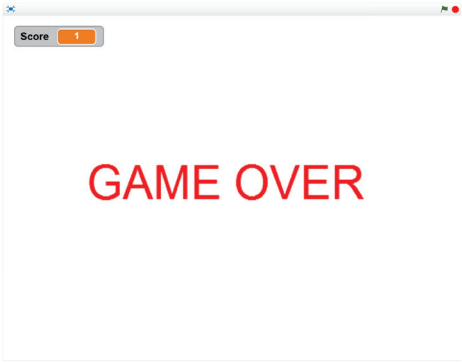
This isn't a real plane. And yes, we are assuming that planes can fly at such low altitudes.

The ball will bounce and come to you but it'll be grounded in your vicinity to give you a fair chance at dodging it. Dodge the ball by jumping

The aeroplane will fly straight at you. Dodge it by sliding under it



If you fail to dodge an obstacle, our sprite returns a message confirming the same



It's GAME OVER once an object hits you.

The Running Animation

You should be familiar with designing and importing backdrops and sprites. The sprite for the kid is not available in the library so import it. The basketball, the school backdrop and the plane sprite are present in the library. Change the school backdrop name to "School".

Select the sprite and scripts tab. He is running when he isn't jumping or sliding. Also, we have created a bird flapping animation before (in the bat game). We know that all we have to do is to just switch between costumes as fast as possible. The various costumes for running are shown in sequence:

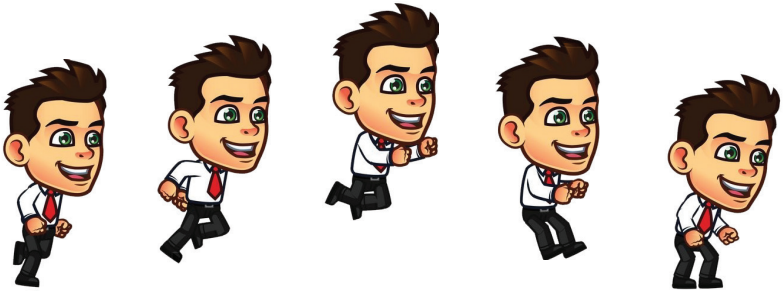
The Running script can easily be create

The X and Y coordinates can be determined by dragging the sprite and dropping it where you require it to be. The glide block automatically reads those coordinates.

Be careful not to place your sprite too much on the lower side or the upper half of the Stage. An ideal position will be the greyest area on the road.

Jumping and Sliding Animation

Let us now write scripts which shall help the boy jump and slide. The costumes for jumping are given below. The 3 costumes in the middle represent time when the boy is in the air while the first and last represent the start and the end of the jump respectively. The 2nd and 4th costumes represent the times it is about to take off and touchdown respectively. While the 3rd costume represents the highest points of the jump.

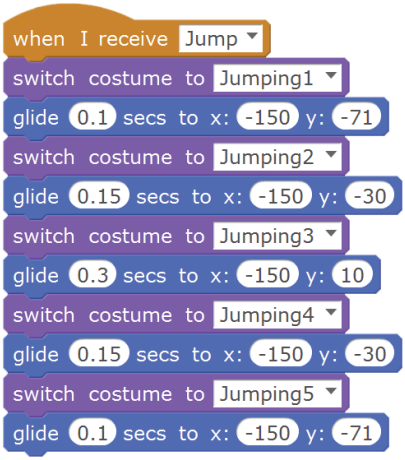


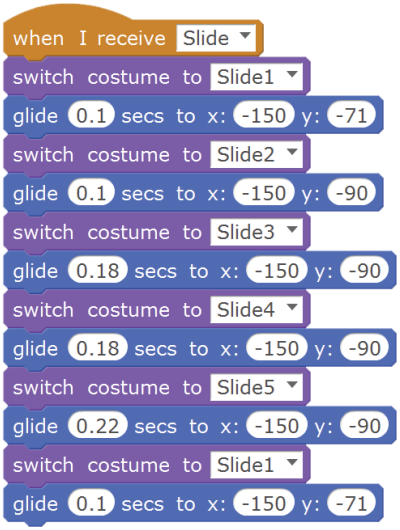
Observe that, and this is important, the sprite should remain in air for more time than on ground because we are jumping.

Another thing worth noting is that when you switch costumes, your Y-coordinate should increase when you switch from 1->2 and 2->3 with the Y-coordinate for the 3rd costume having the highest Y-coordinate (height). To get an idea (of the Y-coordinate), bring your ball sprite near the boy and drag around to see which height is suitable. Make sure that the height is enough so as the boy may be able to go over the ball.

The subsequent images show the stages of sliding. Again, one has to carefully manipulate the time and height on each costume so it may be possible to slide under the plane.

Notice the increased time (in seconds) the sprite spends in costumes 3, 4 and 5, where the boy looks as if he is sliding.





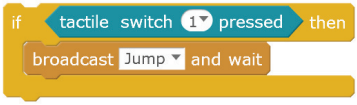
It's OK if you are not able to calibrate the time the sprite spends in each costume as of now. The required time can be adjusted once you run all the scripts and you can actually get an estimate as to how long it will take you to jump over the ball or slide under the plane.

Running, Jumping and Sliding simultaneously

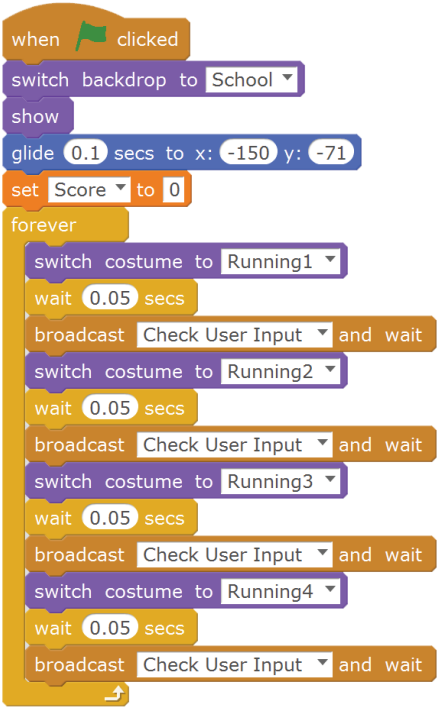
We will jump and slide to avoid obstacles and run at all other times. What we need for this purpose is for the three scripts: "Running, Jumping, And Sliding" to interact with each other. We will make changes in the Running script in such a way that the Jumping and Sliding scripts "know" when to activate. The Jumping and Sliding scripts start off with the "When I receive" block. We will check if the tactile switch 1 or the tactile switch 2 of evive is pressed and then jump or slide accordingly.

We know what to do. The question is: How to do it? Worry not, for Scratch has the required blocks.

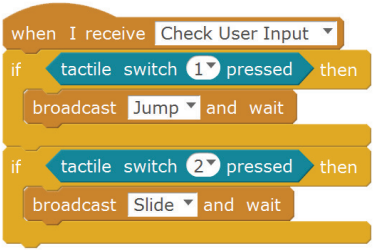
1. When our sprite is in any of the **Running** costumes, we can press the tactile switches to make it jump or slide respectively.
2. In the Robot palette, there is the tactile switch pressed block under evive inbuilt functions extension.
3. We know we have to jump; we should activate the Jumping script now. For this use the if block and then broadcast Jump using the broadcast and wait block.



4. Drag and drop the hat block **When I receive** in **Events** palette. Change the message name to "Check User Input" and snap the above code below it.
5. Where to send the message to check user input? Yeah you guessed it right, as soon as you switch costumes in the **Running** script. The complete running script is shown below. You can observe that you are broadcasting the message after changing costumes. This is done to reduce the lag while playing. You can try the other one, and experience it yourself. Also, to keep track of your performance, create a variable "Score". At the start of the game **Score** should be 0.



6. Try running this and see if the sprite is jumping every time. Similarly, create blocks for sliding. The complete **Check user input** script is shown:



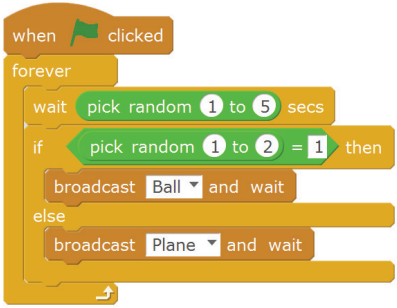
Test whatever you have written until now. Run the scripts and see if the sprite (boy) performs the required actions. This way, debugging is easy. If you write the whole code at once and your program misbehaves, it is very difficult to identify where the mistake may be.

Spawning the ball and the code randomly

It's time for the obstacles now. We are not yet programming our plane and ball. Instead, we are just deciding at what times they will appear on stage. We will pick a random time interval at which the plane and ball appear. See the script given below:

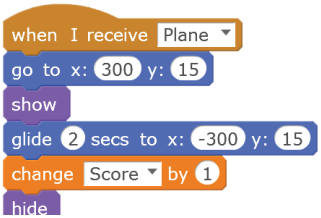
We first wait for a random time period from 1 to 5 seconds. Then, we want any of the ball or the plane to appear so we choose randomly between the two.

Depending on what is broadcasted (chosen), the respective scripts under the ball and the plane sprite get activated. We will now write the scripts for the ball and the plane.



Plane

Starting from the right edge, keep the Y-coordinate constant and **glide** the plane to the same position at the other end. Remember the plane has to disappear at the end. Hence we **hide** it at the end of the script. Also after going to the initial position, we **show** the plane. Every time the plane successfully passes the boy, **Score** should increase by 1. We are executing this using **Change** block in the variable palette.



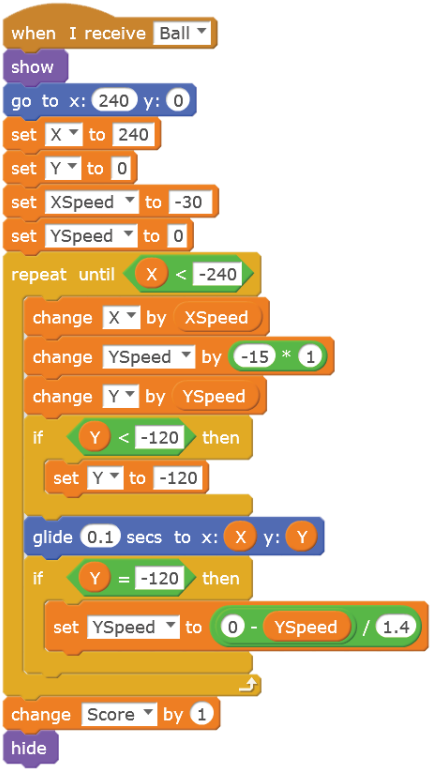
Ball

Next, we write a script for the ball. Start the ball from (240,0). Don't get confused; ground level (where the boy's feet are) is at -120 (Y-coordinate).

Because the ball has to bounce on its' way before settling down, the code is a bit complicated. We create variables X, Y and XSpeed, YSpeed to keep track of position and speeds (in horizontal and vertical directions) respectively.

1. We need the ball to reach the kid, so we keep a constant speed in the x-direction, say 30. Because the speed is towards left, we use a negative sign.
2. The following steps will be repeated till the X-coordinate is less than -240, i.e. leftmost edge:
3. By a speed of -30 we mean the ball will move 30 units to the left every second. So we change the X-coordinate by -30 every time.
4. We want the ball to fall down from height initially but bounce back once it touches ground. However, we do not want it to rise to the same height, i.e. the ball's vertical (Y) speed should decrease once it bounces.

5. For the ball to fall down we should give it speed in the negative Y direction (downward) and change the Y coordinate accordingly.
6. However, we do not want the same to happen once the ball touches the ground, i.e. Y= -120. We give it an upward speed lesser than the speed it fell with (hence the division by 1.4).
7. Just to make sure that the ball doesn't fall below -120, we again set its' Y coordinate to -120 as soon as that happens.
8. Every time the ball successfully passes the boy, Score should increase by 1.
9. We also want to hide the ball at the end of the script and show it at the starting of the script.



Game Over

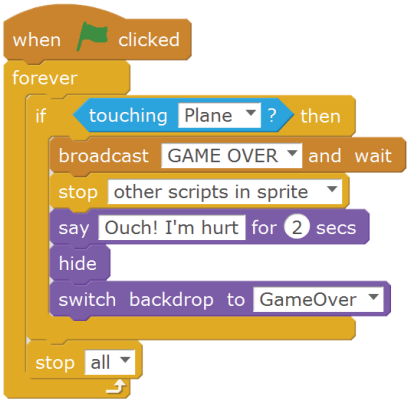
In this section, we will control what happens when the game is over, i.e. either the plane or the ball has struck the boy. When the game ends, we want these things to happen:

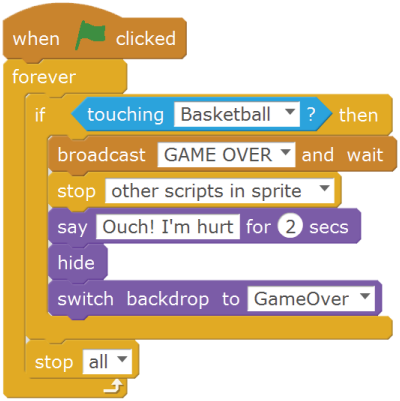
1. The boy displays a message saying he is hurt
2. All motion on the **school** backdrop stops.
3. The backdrop switches to **GAME OVER**

Script for the Boy

We will check if the boy touches the plane or the ball. After that we will broadcast **GAME OVER** throughout the scripts. This will notify the other sprites in the game to execute their respective "GAME OVER" scripts. We will also make the kid say – "Ouch! I'm hurt", hide the sprite and switch backdrops.

We use "stop other scripts in sprite" to avoid glitches. This option is available in the drop down of the stop block.

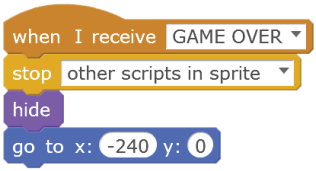


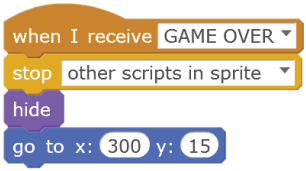


The ball and the plane

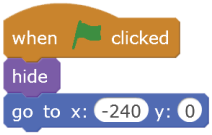
As soon as the ball receives GAME OVER, it stops other scripts, hides the ball and sends it to its' starting point: (240, 0).

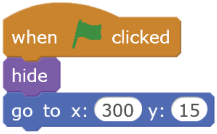
The same script is written for the plane too, except that its' starting point is different from that of the ball.





Also, when you start the game, the sprites, the ball and the plane, should go to their initial position and should be invisible. Given below is the script for the same.

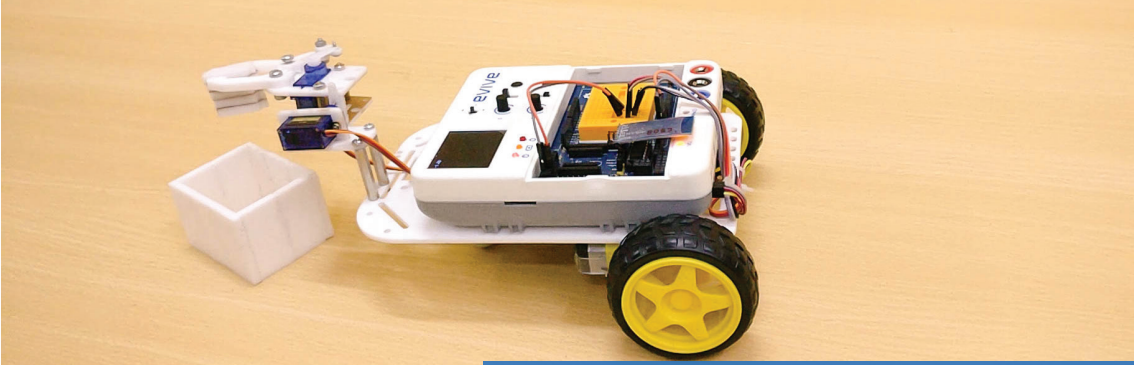




And the game is complete and ready for playing. You can adjust the speed of the moving objects to make the game easy or hard.

Conclusion

In last two chapters, we have built two games. You also learned about some animations and what all things you have consider while making games. In the coming chapters, we will learn to make electronics and robotics projects with evive and program it using Scratch.



CHAPTER 11

Pick and Place Robot

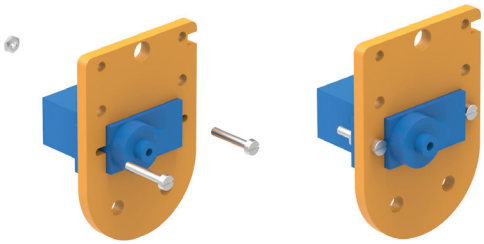
This is the last project on mobile robots. With the component given, you can build three different configurations for gripper and place it on the robot. And Viola! You have a pick and place robot with which you can create small mission and live your dream.

Assembly

The assembled robot will look like the following figure:

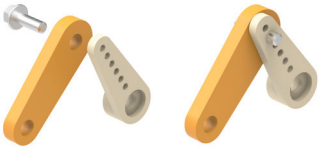
We already know how to assemble a basic mobile robot, i.e. – wheels, Castor, chassis, motors and evive. We will continue our assembly from here. However, first we will assemble our gripper which will help us in grabbing objects and moving them.

1. Insert the micro servo motor into the slot into the Gripper Plate shown below.
2. Fasten the servo motor to the Gripper Plate using M2 bolts of length 12mm.



Steps 1 and 2

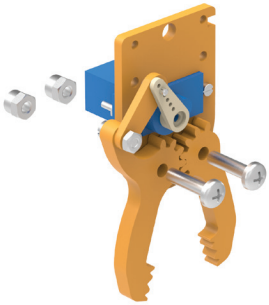
3. As shown in the figure, lock the Gripper Link with the one sided servo horn (you get with the micro servo) using an **M2 screws** (can be found with the servo horn). Do not tighten it too hard, keep it loosen so that it can move.



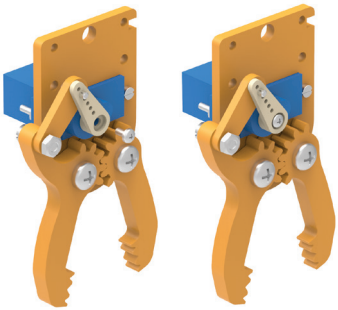
Step 3

4. Affix this assembly to the Gripper Claw 1 (see the figure to know which one) using an **M3 bolt of 12mm length** and **M3 nut**.

Step 4

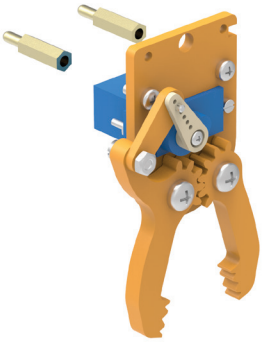


Step 5

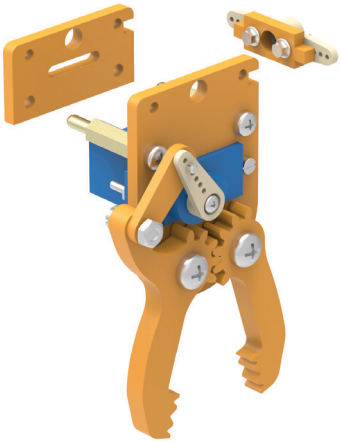


Step 6

5. Bring this assembly to the Gripper Plate-Servo assembly such that the free Servo head locks into the motor. Also, bring along the other Gripper claw and fasten both the claws to the plate with **M4 bolts of 16mm length** and **M4 nuts**.
6. Lock the horn to the head using the **M2 screw** provided with the servo accessories. Using **evive control menu**, set the servo angle to 45 degrees.
7. To the free ends of the bolts which hold the Servo, attach 15mm standoffs.

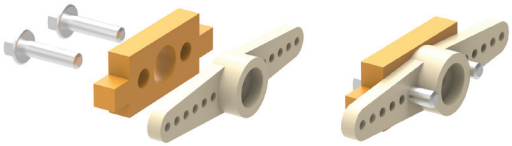


Step 7



Step 9

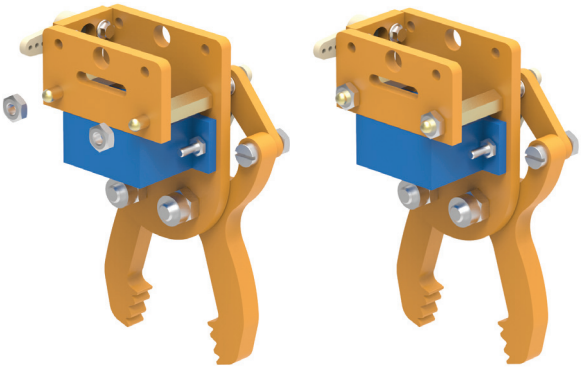
8. Attach a micro Servo horn to the part given below using self-threading **M2 screw** provided with servo accessories.



Step 8

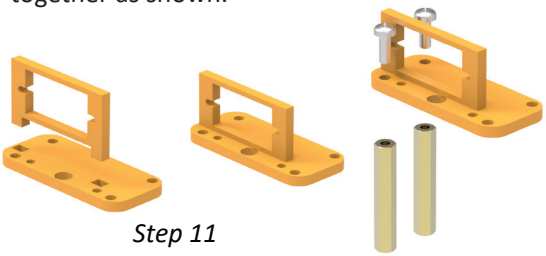
9. Arrange the gripper parts below in the configuration shown. Notice the slots in the larger parts and the protrusions (sideways) of the smaller part (to which the horn is fixed).

10. Fasten at the free end of each standoff using an **M3 nuts.**

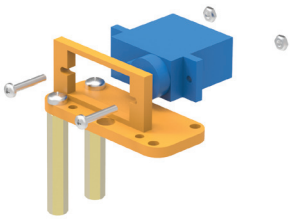


Step 10

11. Now, we will start assembling the second servo in its slot. Attach the parts together as shown.

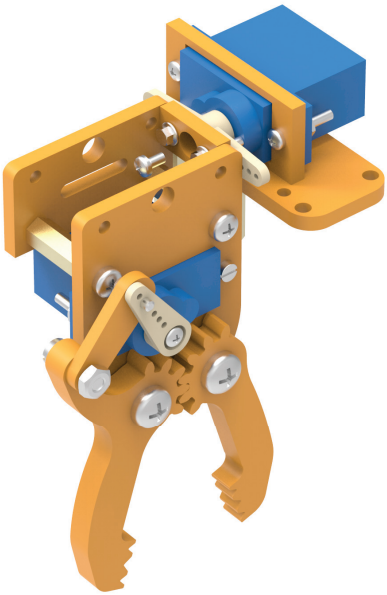


Step 11



Step 12

12. Push the servo into the slot and attach **30mm standoffs** as shown above using **M3 bolts of 8mm length**.
13. Fasten the Servo to the frame using **M2 bolts** and **M2 nuts**.
14. There is a free Servo horn on the previous assembly and a free head on this one. Now lock them together using the bolt that came with servo accessories. Set the servo angle to 0 degrees first using evive.



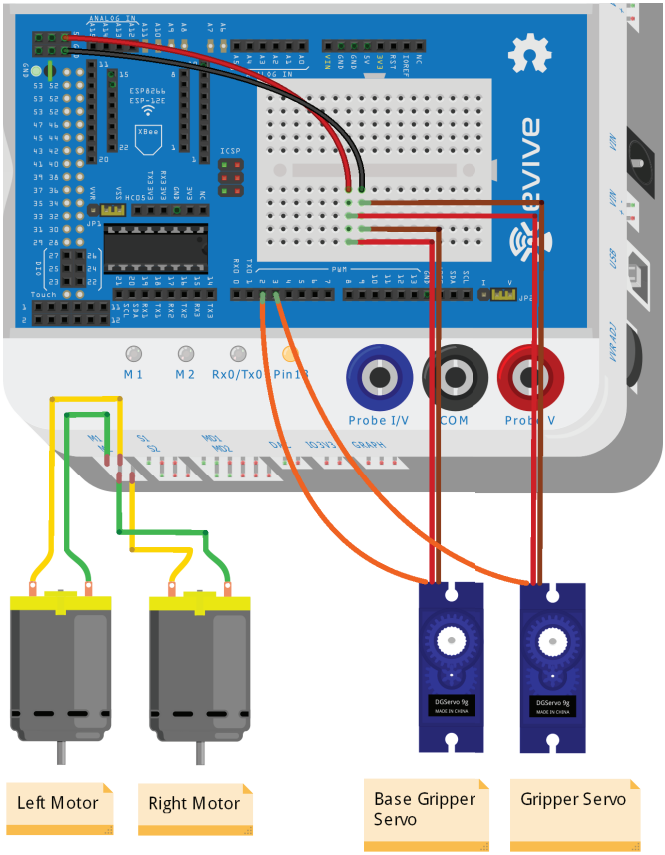
Step 13

If you do not set the base servo angle to 0 degree and gripper servo angle to 90 degrees, your robot will not work properly. Take your time and do this properly

15. Attach the completed assembly at the front of the chassis through the free ends of standoffs using **M3 bolts of 8mm length**.

And the Pick and Place Mobile Robot is ready.

Circuit Diagram



Logic and Flow Chart

Your Smartphone App send code to evive via Bluetooth. Thus, according to the data received, the robot takes actions which are:

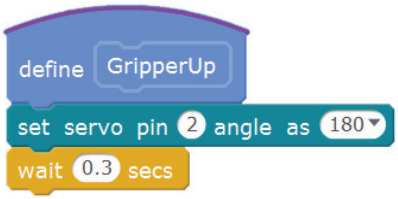
- Go Straight
- Go Backward
- Turn Right
- Turn Left
- Brake
- Gripper Up
- Gripper Down
- Gripper Close
- Gripper Open

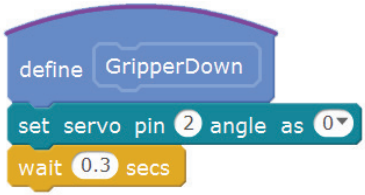
Flow chart is same as the Smartphone controlled robot. Just you have to add few more if-else conditions for Gripper Up (Received_Code = 5), Gripper Down (Received_Code = 6), Gripper Close (Received_Code = 7) and Gripper Open (Received_Code = 8).

Scratch Scripts

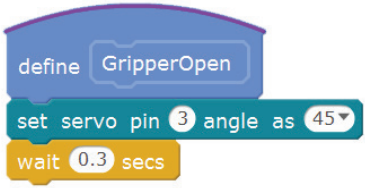
Follow the steps 1 to 14 of Smartphone Controlled Robot. We will now proceed further:

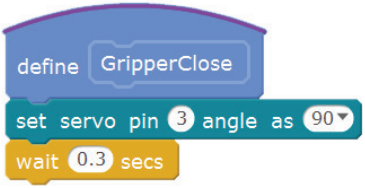
1. In Initialization definition, add blocks to set both servos to 0 degrees.
2. Make blocks for moving gripper up and down. Remember for up position servo angle is 180 degree and for down position it is 0 degree. Add some delay for smooth functioning.





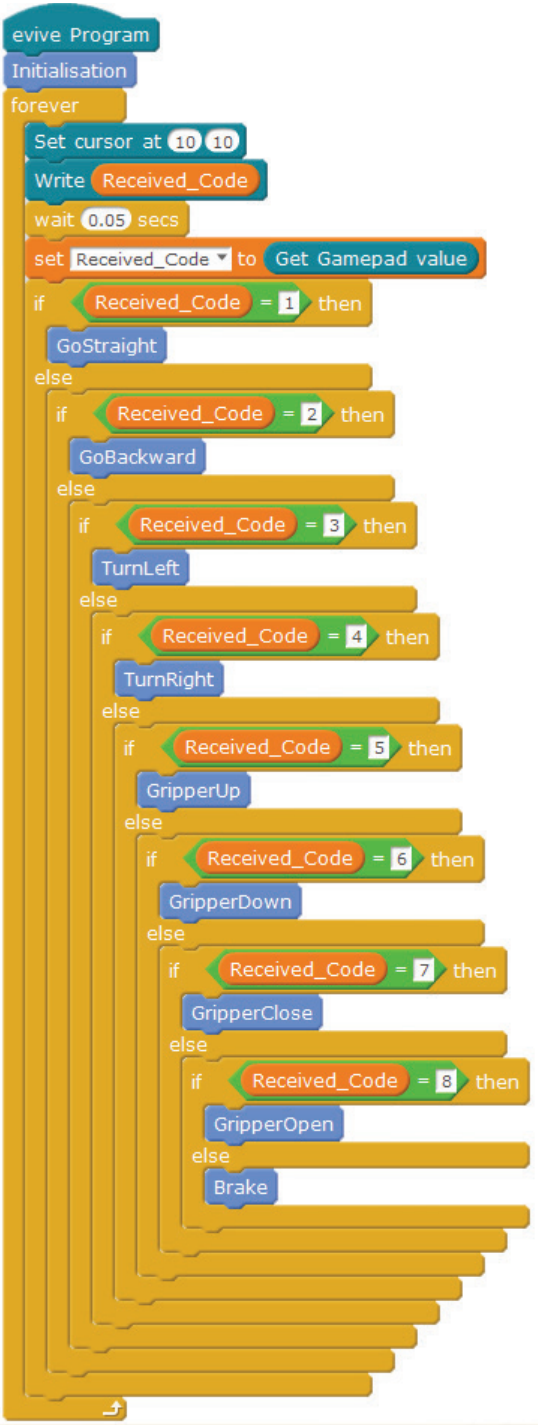
3. Similarly make blocks for opening and closing gripper. Remember, for opening the gripper, servo angle should be 45 degree and for closing, it should be 90 degrees.





4. Coming to the main code using **if-else** block in control palette and the flow chart make the complete script.
5. After completing the main script, upload the Arduino code generated into evive and run the robot.

Enjoy playing with your robot.


evive Program
Initialisation
forever
Set cursor at 10 10
Write Received_Code
wait 0.05 secs
set Received_Code to Get Gamepad value
if Received_Code = 1 then
GoStraight
else
if Received_Code = 2 then
GoBackward
else
if Received_Code = 3 then
TurnLeft
else
if Received_Code = 4 then
TurnRight
else
if Received_Code = 5 then
GripperUp
else
if Received_Code = 6 then
GripperDown
else
if Received_Code = 7 then
GripperClose
else
if Received_Code = 8 then
GripperOpen
else
Brake

Explore more Robots

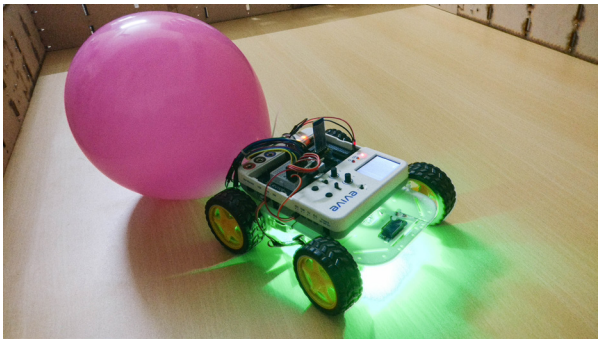
In previous four chapters, you have made five varieties of mobile robot. There are more number possible robots which you can make with the starter kit. We are mentioning few more robots:

Light Follower Robot

The light follower robot is another autonomous robot which you can build with the starter kit. This robot uses photoresistors to sense light at the front, right side, left side and at the back of the robot. If any of the sensor senses more light, it starts navigating towards that direction. For example, if the robot senses more light at the back of the robot, then the robot turns by 180 degrees.

Balloon Popping Robot

This robot have a needle attached to the servo horn (same as the obstacle avoidance robot except ultrasonic sensor is replaced by needle). The robot will be controlled by mobile App. You can control the orientation of needle (pointing straight or sidewards).



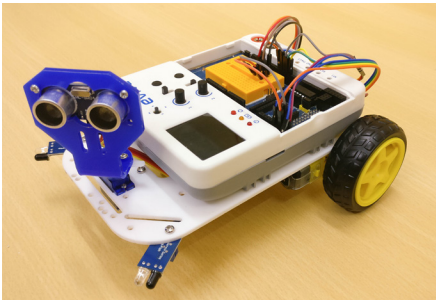
If you have multiple robots you can also have a match where you have to pop the balloon of the other opponent before he pop your balloon.

Robo-Soccer

This a team battle, you can make 2 or 4 robots, and have a curved sheet attached at the front of the robot, which will help the robot to move a ball. Just like a soccer match there will be two goal post. Design your robot and arena, bring you friends and score goals.

Advanced Obstacle Avoiding Robot

If you have made an obstacle avoiding robot, then you must have realized that many time the robot crashes to the obstacle because the it was either on the left or the right side of the robot and not in the vicinity of the ultra-sonic sensor. Thus using the IR sensors, we can detect those sensors and avoid collision. Given below is the image of the robot.

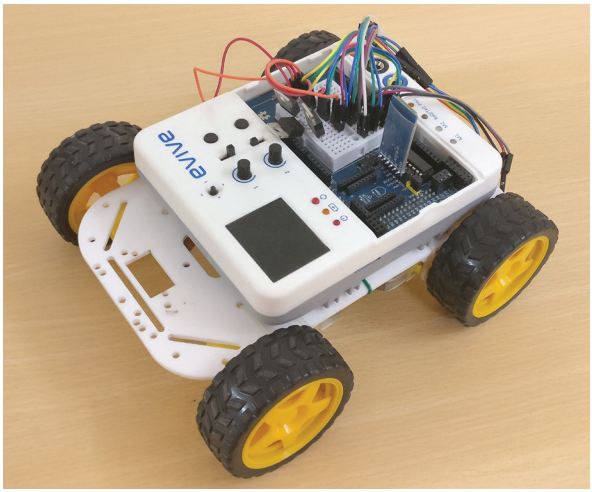


Scorpion Robot

This robot is another pick and place robot which have the gripper in the horizontal plane. This robot grips objects just like a Scorpion.

Four Wheel Drive Robot

If you have two more BO motors and wheel (easily available in electronics shops), then you can build a four wheel drive robot. The base plate have holes for mounting two more motors.



evive Scratch Blocks

Arduino Extension

Just like any hat block, its' purpose is to start off a script in a Scratch program in Arduino Mode.

This block reads the state of the digital pin input by the user and hence checks if the digital pin is "HIGH" or "LOW".

This block reads the analog value (from 0-1023) from an analog pin on the board.

This block returns the time since the timer has been running.

This block resets the timer, i.e. value read by the timer block is set to 0.

This block initializes a state (HIGH or LOW) to the specified digital pin.

This block sets the PWM (square wave) at the specified PWM pin.

This block sets the servo motor at a specified angle depending on which pin the servo is connected.

This block writes transfers of data from your device (PC) to evive (or Arduino board)

This block returns the number of bytes available at Serial

This block reads binary data from the serial port.

This block play a tone from a piezo buzzer (46 for evive).

This block returns the distance (float) from an ultrasonic sensor.

evive Inbuilt Functions

This block is used to start every program in Scratch in Arduino Mode

This block uploads evive firmware from which user can use menu based system to accomplish various simple tasks.

This block checks if the specified tactile switch on evive is pressed.

This block returns the analog value of the specified potentiometer of evive (from 0-1023).

This block returns “True” if the navigation key is pressed in z direction, else “False”.

The block returns the value of “hours”, “minutes” or “seconds” from evive’s RTC.

The block returns the value of “day”, “Month”, “Year” or “Weekday” from evive’s RTC.

This block returns “True” if the specified touch input is touched, else “False”.

This block locks (more torque than free motor) the motor according to the motor port input by the user.

This block frees the motor according to the motor port input by the user.

This block checks whether the slide switch (1 or 2) is in state (1 or 2). If yes, it returns “True”, else “False”.

This block checks if the navigation key is in one of the four particular directions. (1 is for up, 2 for Right, 3 for Down and 4 for Left).

This block takes the motor channel, direction of rotation and speed of rotation as input from the user and moves the motor accordingly.

This block sets the time on evive's Real Time Clock (RTC) to the time you specify in the input.

This block set the date and weekday on evive's Real Time Clock (RTC)

evive TFT Display Extension

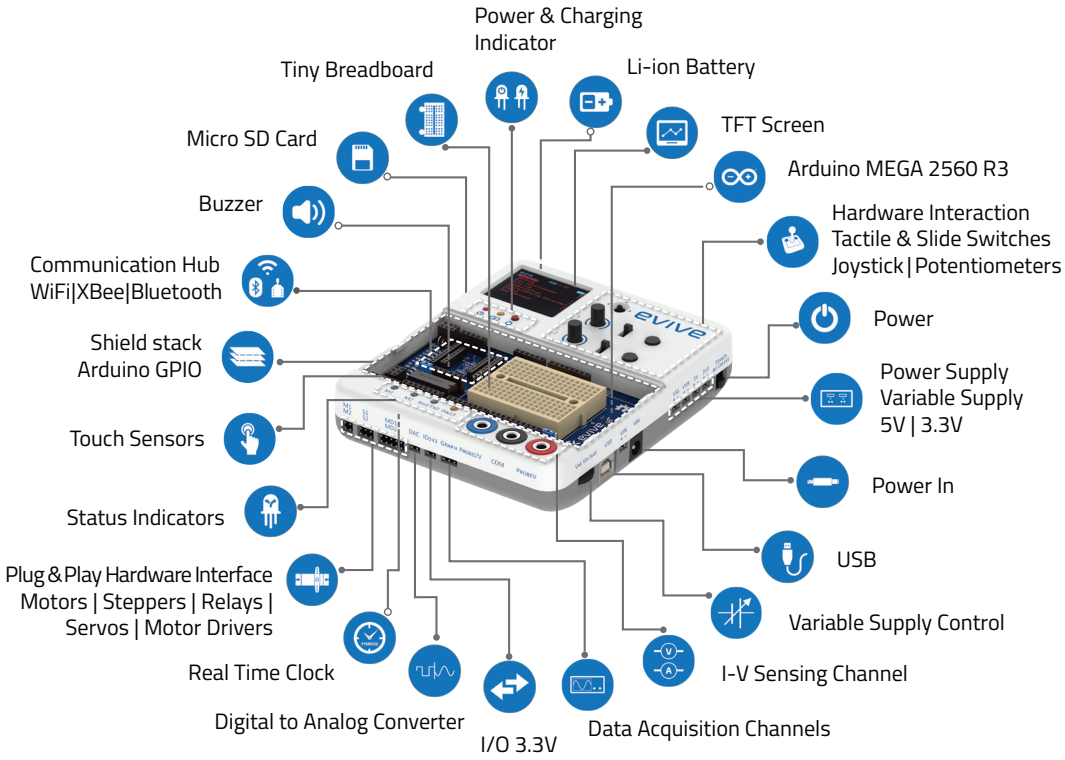
This block changes the color of the entire screen. Select suitable color from the drop down available.

It sets the cursor at the specified coordinate. Origin (0,0) is at the top left corner of the screen. Positive X-direction is to right while positive Y-direction is downward.

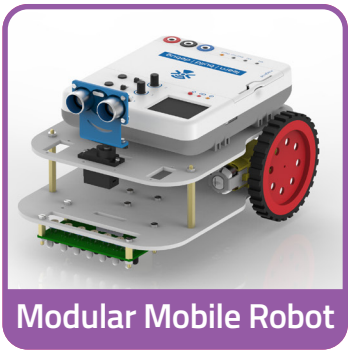
This block changes the text size. The size ranges from 1-6 and can be chosen from the drop down on this block.

This block is used to write text on evive's TFT display.

evive | Features

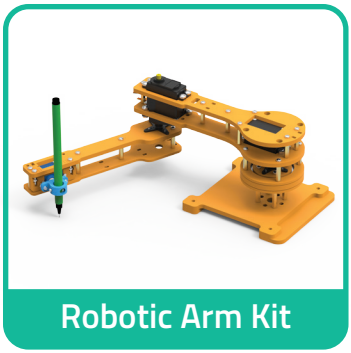


Other DIY Learning Modules



Modular Mobile Robot

Home Automation Kit



Robotic Arm Kit

Agilo Reseach Private Limited
Ahmedabad, Gujarat, India - 380009
contact@evive.cc
+91-8726 533 960, +91-8765 696060
Explore more at learn.evive.cc