Topic	ROBOTIC ARM 4	
Class Description	**Students will learn how to attach a camera node to the UR5e robotic arm and create an object sorting system. They will also learn how to add different textures to solid nodes.**	
Class	**PRO C291**	
Class time	**50 mins**	
Goal	● Attaching a camera node. ● Object detection and recognition using a camera. ● Sorting objects based on their color. ● Adding textures.	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher -Led-Activity 1** **Student-Led Activity 1** **Wrap-Up**	**5 mins** **20 mins** **20 mins** **20 mins**
Credit & Permissions:	**This project uses Webots, an open-source mobile robot simulation software developed by Cyberbotics Ltd.** **License**	
WARM-UP SESSION - 10 mins		

Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** ● Greet the student. ● Revision of previous class activities. ● Quizzes.	**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides

WARM-UP QUIZ
Click on In-Class Quiz

Activity Details

Following are the session deliverables:

- Appreciate the student.
- Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.

TEACHER-LED ACTIVITY - 20 mins

Teacher Initiates Screen Share

ACTIVITY

- **Adding multiple solid boxes.**
- **Adding camera node to the UR5e robotic arm.**

Teacher Action	Student Action
Do you remember what we did in the last class? Great, if you have any doubts from the last class, please ask. *Note : Teacher will clear the doubts, if students have any.*	**ESR :** Yes, we added a distance sensor to our robotic arm.

If there are no more questions from the previous classes, let's jump to today's class.

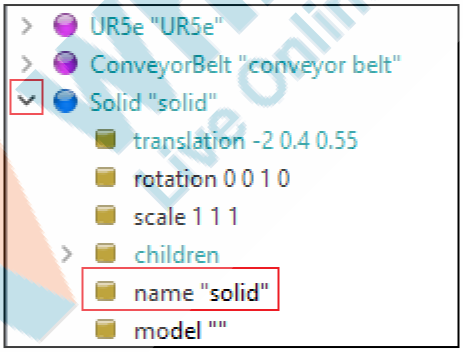
Now we have learnt how to pick and drop objects using the UR5e robotic arm. It is time to make the object sorting system we initially decided. To create the sorting system, first we should have different objects on our conveyor belt.

We already have a red colored solid on the conveyor belt, let's make a **green colored solid**.

For that, let's first open the teacher boilerplate link and download all the files from it. Open all the downloaded files in the webots software.

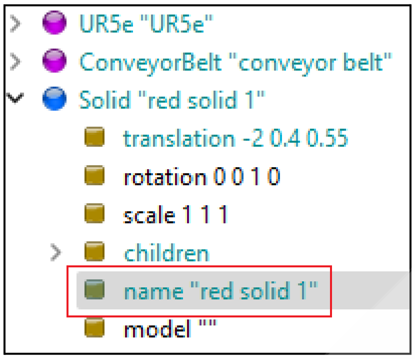
Now, creating a similar solid box is very simple. We can simply **copy and paste** the **red colored solid box** and then change its color.

For that, let's first **expand** the **red colored solid node** and **rename** it to avoid any confusion.



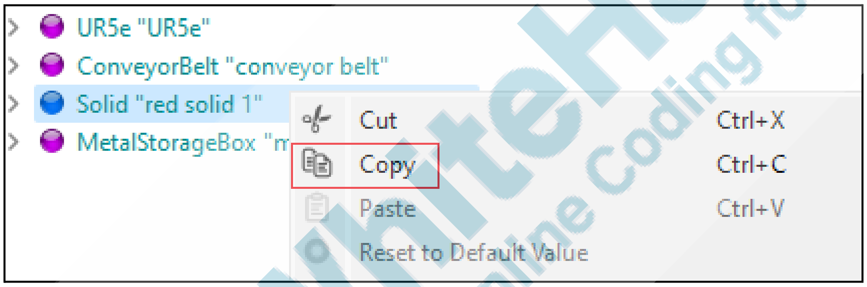
```
> UR5e "UR5e"
> ConveyorBelt "conveyor belt"
v Solid "solid"
    translation -2 0.4 0.55
    rotation 0 0 1 0
    scale 1 1 1
  > children
    name "solid"
    model ""
```

To **rename** it, change the **name** property to **'red solid 1'**.

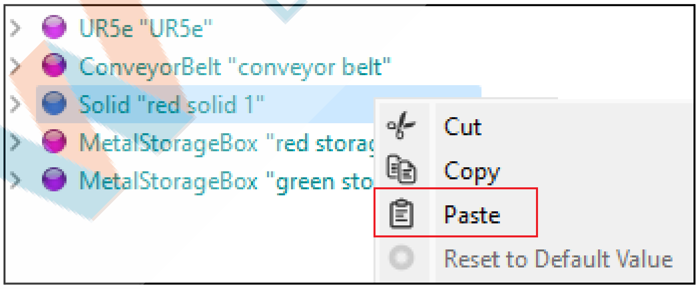


Next, close the properties of the **'red solid 1'** node and **right** click on it. You will see the **Copy** option. Click on it.

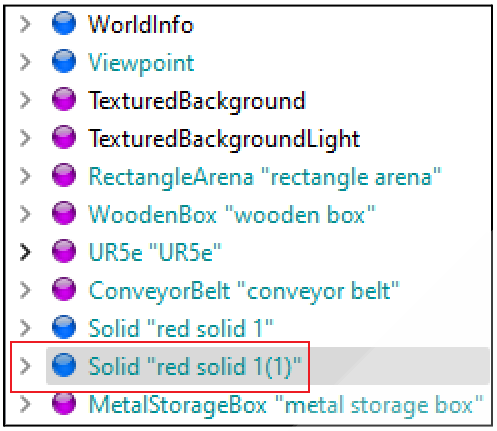
Your node is **copied**.



To **paste** it, right click on it again and click on the **Paste** option.

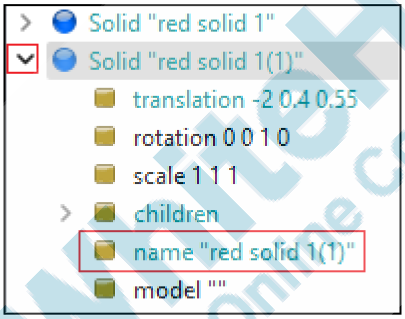

UR5e "UR5e"
ConveyorBelt "conveyor belt"
Solid "red solid 1"
MetalStorageBox "red storag
MetalStorageBox "green sto
Cut
Copy
Paste
Reset to Default Value

You will see that a **duplicate solid node** named **'red solid 1(1)'** will be created.



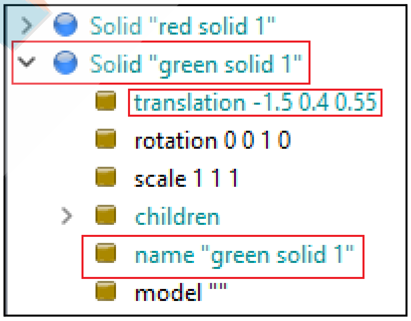
WorldInfo
Viewpoint
TexturedBackground
TexturedBackgroundLight
RectangleArena "rectangle arena"
WoodenBox "wooden box"
UR5e "UR5e"
ConveyorBelt "conveyor belt"
Solid "red solid 1"
Solid "red solid 1(1)"
MetalStorageBox "metal storage box"

Let’s **expand** the new copied node and change its properties.



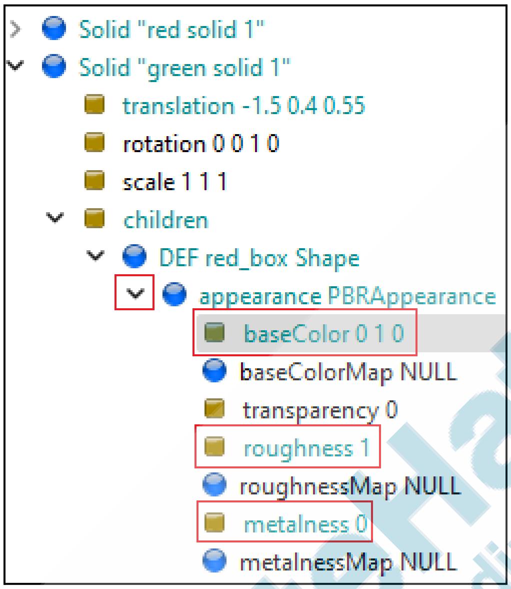
Solid "red solid 1"
Solid "red solid 1(1)"
translation -2 0.4 0.55
rotation 0 0 1 0
scale 1 1 1
children
name "red solid 1(1)"
model ""

First, let’s change its **name** to **‘green solid 1’**. Next change its **translation** to **-1.5** in **x direction, 0.4** in **y direction** and **0.55** in **z direction**, to set it apart from the original ‘red solid 1’ node.



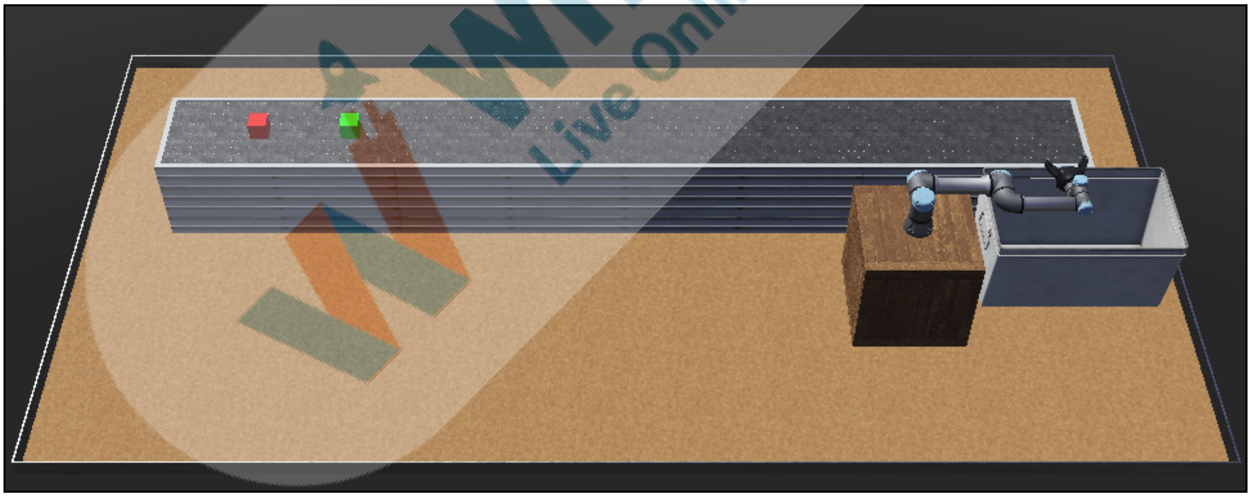
Solid "red solid 1"
Solid "green solid 1"
translation -1.5 0.4 0.55
rotation 0 0 1 0
scale 1 1 1
children
name "green solid 1"
model ""

Also, expand the **appearance** property and change the **base color** to **0** in **red, 1** in **green** and **0** in **blue.** Also make sure that **roughness value** is set to **1** and **metalness** value to **0**.

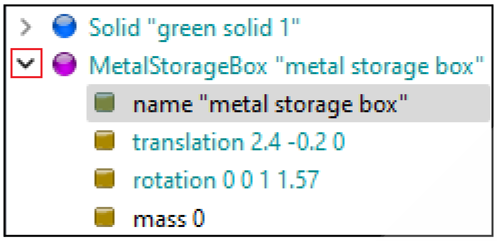
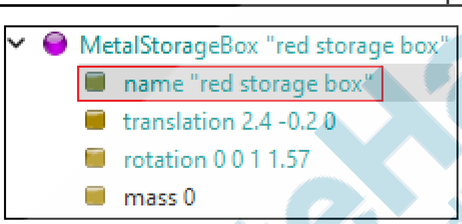
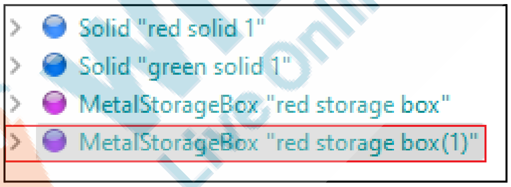


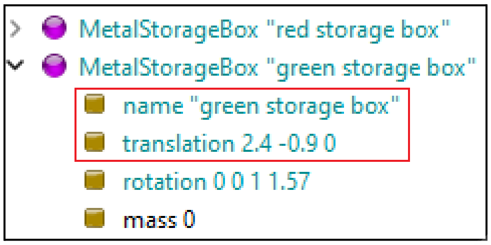
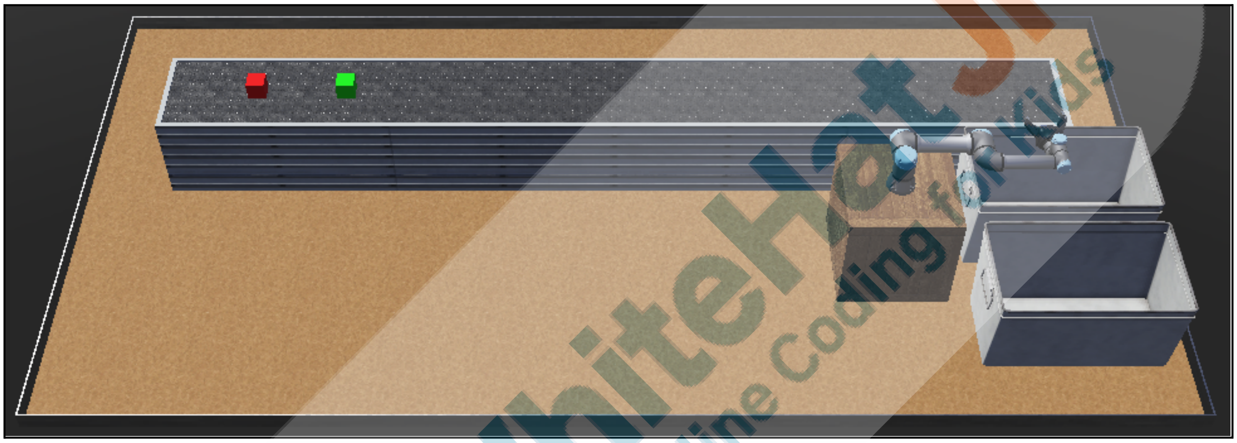
```
> Solid "red solid 1"
v Solid "green solid 1"
    translation -1.5 0.4 0.55
    rotation 0 0 1 0
    scale 1 1 1
  v children
    v DEF red_box Shape
      v appearance PBRAppearance
          baseColor 0 1 0
          baseColorMap NULL
          transparency 0
          roughness 1
          roughnessMap NULL
          metalness 0
          metalnessMap NULL
```

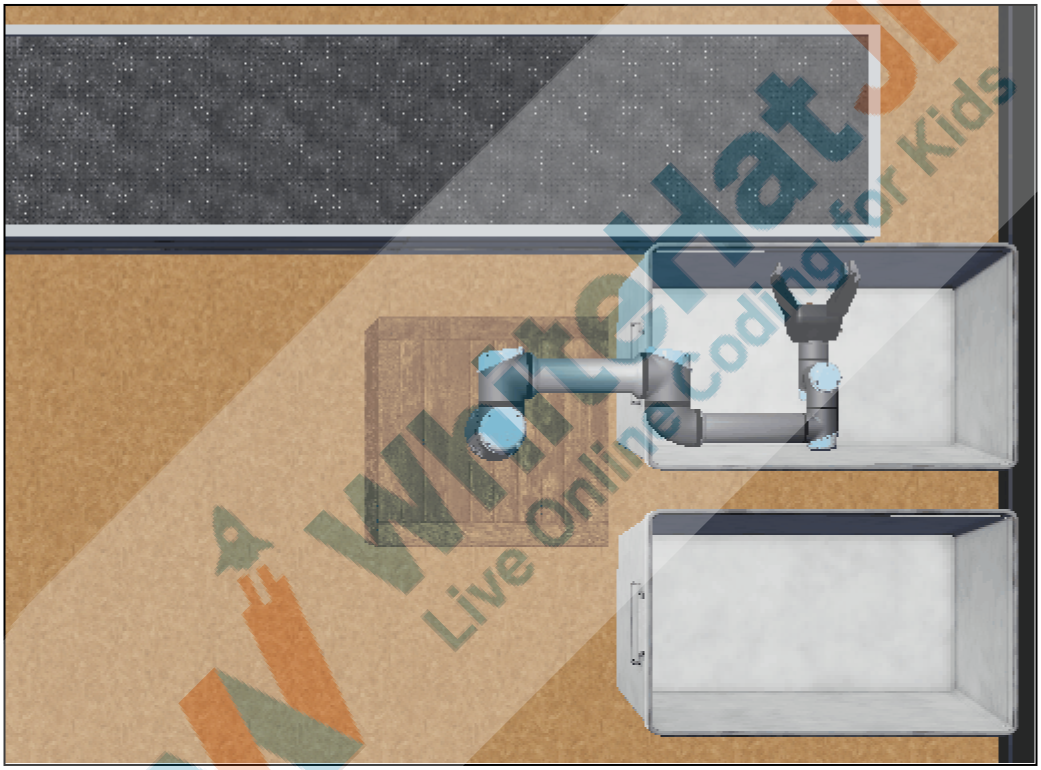
Save your work till here. Your environment will look as shown in the graphic.

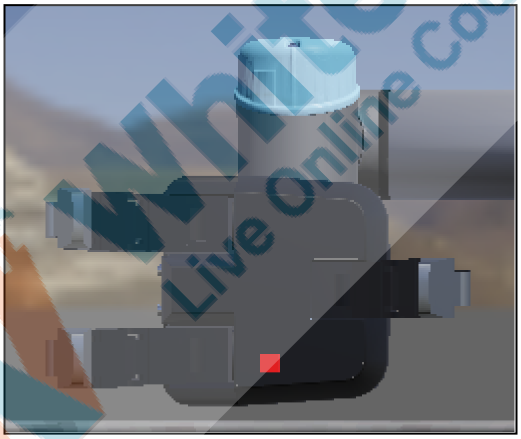


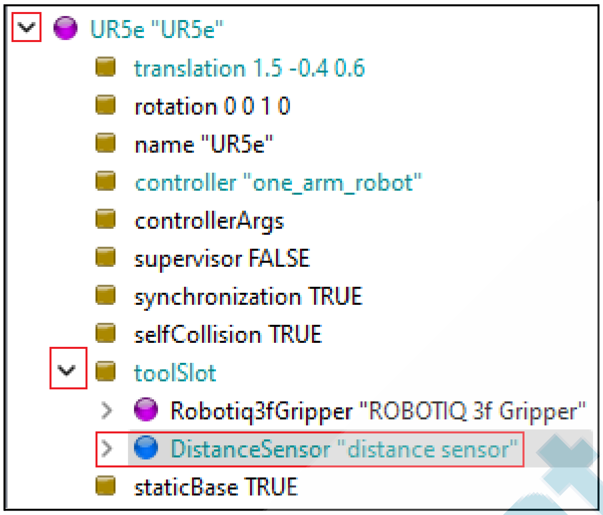
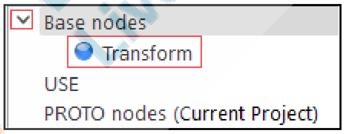
Next, after sorting both the solids, they have to be put into different metal storage boxes. So let’s create a new one, by **copying** the old one.

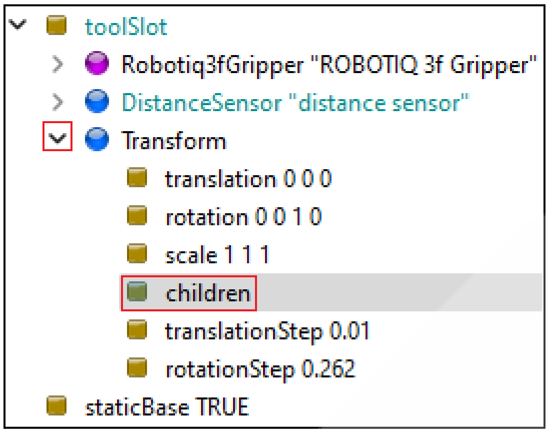
Let's repeat the same process by first expanding the metal storage box node.	
 Solid "green solid 1" MetalStorageBox "metal storage box" name "metal storage box" translation 2.4 -0.2 0 rotation 0 0 1 1.57 mass 0	
Let's change its name to **'red storage box'**. After that, **copy** the red storage box node and **paste** it.	
 MetalStorageBox "red storage box" name "red storage box" translation 2.4 -0.2 0 rotation 0 0 1 1.57 mass 0	
A duplicate of the storage box node named **'red storage box(1)'** will be created.	
 Solid "red solid 1" Solid "green solid 1" MetalStorageBox "red storage box" MetalStorageBox "red storage box(1)"	
Let's change some properties of the newly created metal storage box node. For that, a) First let's **expand** the node. b) Change its name to **'green storage box'** node. c) Change its **translation** to **2.4** in **x direction, -0.9** in **y direction** and **0** in **z direction**.	

 MetalStorageBox "red storage box" MetalStorageBox "green storage box" name "green storage box" translation 2.4 -0.9 0 rotation 0 0 1 1.57 mass 0	
Save your work till here. Your environment will look as shown in the graphic below.	
	
Let's play the simulation and see what the output looks like. Ok, there is one more challenge. Can you tell me what it is?	**ESR** : Though we have created different metal storage boxes for each of the colored solid boxes, it is still putting all the solid boxes in the same container.
Correct, can you tell me the reason for this ?	**ESR** : I think there is no way for the robotic arm to detect the difference between a red solid block and a green solid block.
That's correct. Awesome.	

Can you tell me how we can detect the difference between the blocks? We can actually add a camera to our robotic arm and by using the object detection and recognition algorithms, we can actually differentiate between the colored boxes. Does it sound exciting ?	**ESR** :Varied **ESR** : Yes
	
Ok, there is one more challenge. Can you tell me what it is? Correct, can you tell me the reason for this ?	**ESR** : Though we have created different metal storage boxes for each of the colored solid boxes, it is still putting all the solid boxes in the same container. **ESR** : I think there is no way

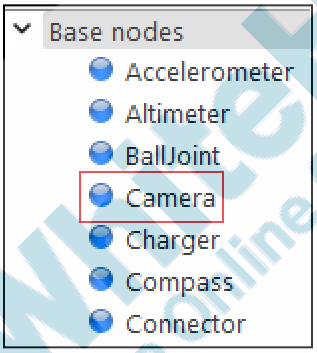
That’s correct. Awesome. Can you tell me how we can detect the difference between the blocks? We can actually add a **camera** to our robotic arm and by using the **object detection** and **recognition algorithms**, we can actually differentiate between the colored boxes. Does it sound exciting ? Great, let’s attach a camera on the top of our gripper palm, opposite to the distance sensor node.	for the robotic arm to detect the **difference** between a **red solid block** and a **green solid block**. **ESR** :Varied **ESR** : Yes
	
To do so, a) **Expand** the **UR5e** node. b) Expand the **toolslot** property. c) Click on the **distance sensor node once**, so that our camera node comes right under it. d) Click on the **add object button**.	

 UR5e "UR5e" translation 1.5 -0.4 0.6 rotation 0 0 1 0 name "UR5e" controller "one_arm_robot" controllerArgs supervisor FALSE synchronization TRUE selfCollision TRUE toolSlot Robotiq3fGripper "ROBOTIQ 3f Gripper" DistanceSensor "distance sensor" staticBase TRUE	
Expand the **base nodes** and select the **transform** node. Can you tell me why we have added the transform node first? We have added a transform node in order to rotate our camera in a very easy manner, such that the axes of the camera are oriented properly. This statement will make much more sense when we will adjust our camera later in the lesson.	ESR : Varied
 Base nodes Transform USE PROTO nodes (Current Project)	
To add a camera node, a) Let's first expand the **transform** node. b) **Double click** on the **children** property.	



```
toolSlot
  Robotiq3fGripper "ROBOTIQ 3f Gripper"
  DistanceSensor "distance sensor"
  Transform
    translation 0 0 0
    rotation 0 0 1 0
    scale 1 1 1
    children
    translationStep 0.01
    rotationStep 0.262
staticBase TRUE
```

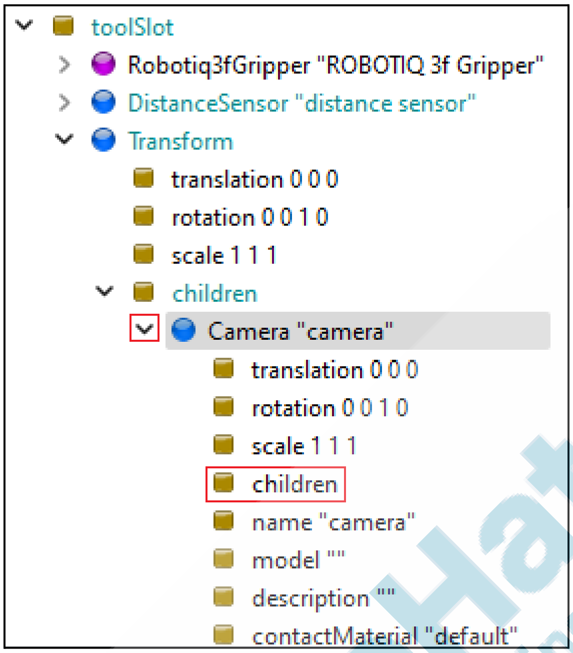
A window will appear. Expand the **base nodes** and **double click** on the **Camera** node.



```
Base nodes
  Accelerometer
  Altimeter
  BallJoint
  Camera
  Charger
  Compass
  Connector
```

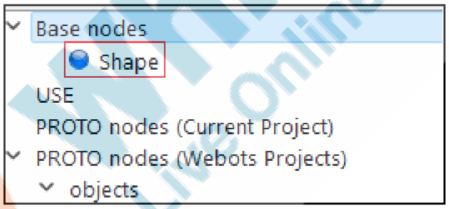
You will see that a camera node is added under the **children property** of the **transform** node.

Let's give a **shape** to our camera node. For that, let's **expand** it first and **double click** on the **children** property of the **camera** node.



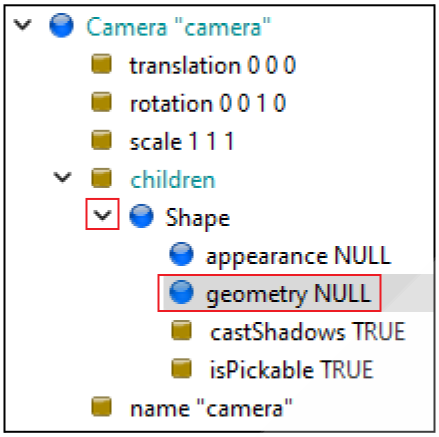
toolSlot
Robotiq3fGripper "ROBOTIQ 3f Gripper"
DistanceSensor "distance sensor"
Transform
translation 0 0 0
rotation 0 0 1 0
scale 1 1 1
children
Camera "camera"
translation 0 0 0
rotation 0 0 1 0
scale 1 1 1
children
name "camera"
model ""
description ""
contactMaterial "default"

A window will appear. Expand the **base nodes** and double click on the **shape** node.



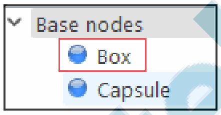
Base nodes
Shape
USE
PROTO nodes (Current Project)
PROTO nodes (Webots Projects)
objects

To give our camera a shape, **expand** the **shape** node and double click on the **geometry** property.



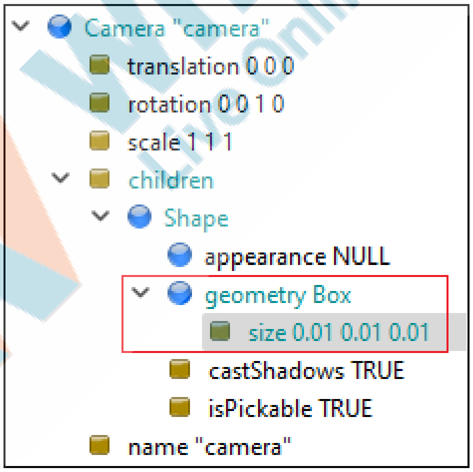
```
Camera "camera"
    translation 0 0 0
    rotation 0 0 1 0
    scale 1 1 1
    children
        Shape
            appearance NULL
            geometry NULL
            castShadows TRUE
            isPickable TRUE
    name "camera"
```

Expand the **base nodes** and double click on the **box** node. Your camera node will appear with a very big box shape.



```
Base nodes
    Box
    Capsule
```

To **resize** your camera, expand the **geometry** property and change its **size** to **0.01 m in x, y** and **z direction**.

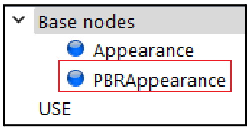


```
Camera "camera"
    translation 0 0 0
    rotation 0 0 1 0
    scale 1 1 1
    children
        Shape
            appearance NULL
            geometry Box
                size 0.01 0.01 0.01
            castShadows TRUE
            isPickable TRUE
    name "camera"
```

Once we have a fixed shape for our camera, let's give it an **appearance**. For that,

a) Double click on the **appearance** node.
b) Expand the **base nodes**.

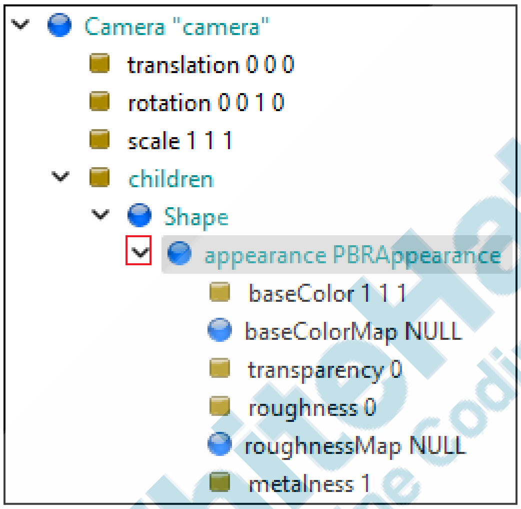
c) Double click on the **PBR Appearance**.



Base nodes
- Appearance
- PBRAppearance

USE

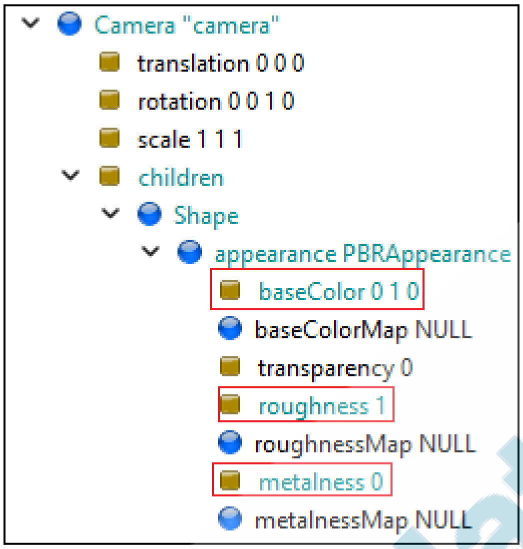
To change the **appearance** properties, expand it first.



```
Camera "camera"
    translation 0 0 0
    rotation 0 0 1 0
    scale 1 1 1
    children
        Shape
            appearance PBRAppearance
                baseColor 1 1 1
                baseColorMap NULL
                transparency 0
                roughness 0
                roughnessMap NULL
                metalness 1
```

Change the **base color** to **0** in **red, 1** in **green** and **0** in **blue.** Also change the **roughness** to **1** and **metalness** to **0**.

Great, our camera will be **green** in color.



Camera "camera"
translation 0 0 0
rotation 0 0 1 0
scale 1 1 1
children
Shape
appearance PBRAppearance
baseColor 0 1 0
baseColorMap NULL
transparency 0
roughness 1
roughnessMap NULL
metalness 0
metalnessMap NULL

Now after we have set the **shape** and **appearance** of the camera, it's time to **orient** our camera properly.

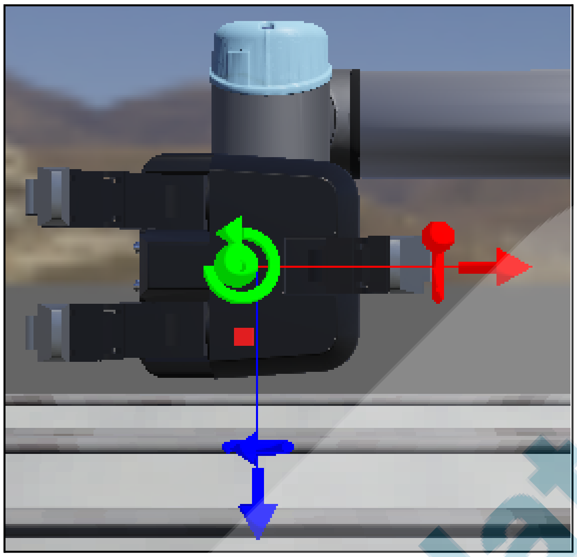
If you look at the graphic given below, you will see that

a) The **x axis** or the **red colored axis** of the camera is pointing in the **right** direction.
b) The **z axis** or the **blue colored axis** is pointing in the **downward** direction.

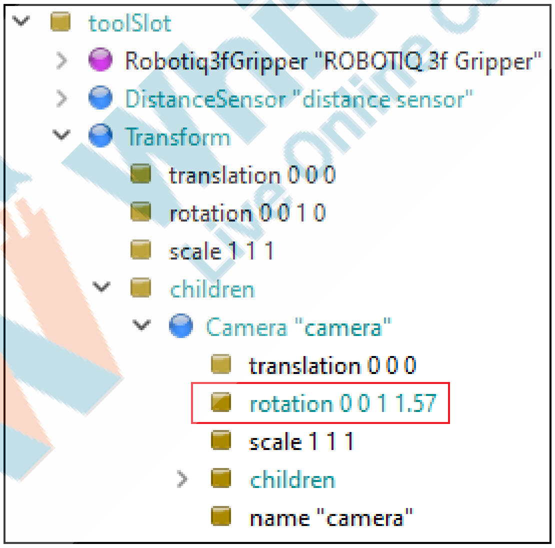
To get a proper video feed we need to make sure that,

a) The **x axis** of the camera should be pointing **outwards**.
b) The **z axis** should be pointing **upwards**.

To achieve this, we need to rotate our camera node.

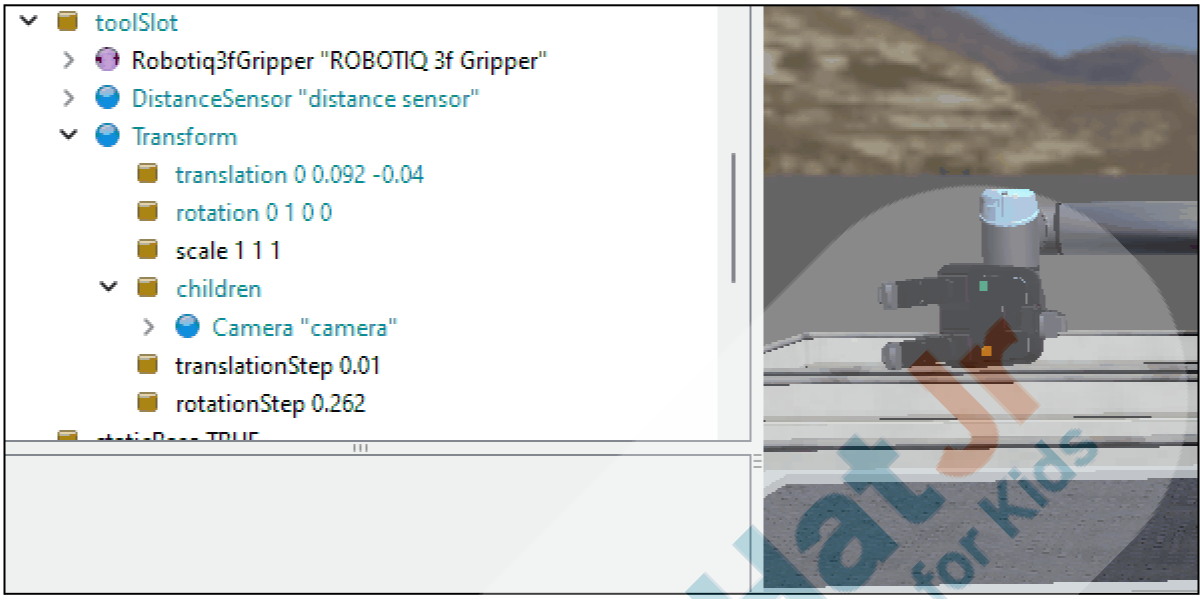


Let’s change the **rotation** property of the camera to **0** in **x direction, 0** in **y direction, 1** in **z direction** and specify the **rotation angle** as **1.57 radians** or **90 degrees.**



```
toolSlot
  Robotiq3fGripper "ROBOTIQ 3f Gripper"
  DistanceSensor "distance sensor"
  Transform
    translation 0 0 0
    rotation 0 0 1 0
    scale 1 1 1
    children
      Camera "camera"
        translation 0 0 0
        rotation 0 0 1 1.57
        scale 1 1 1
        children
        name "camera"
```

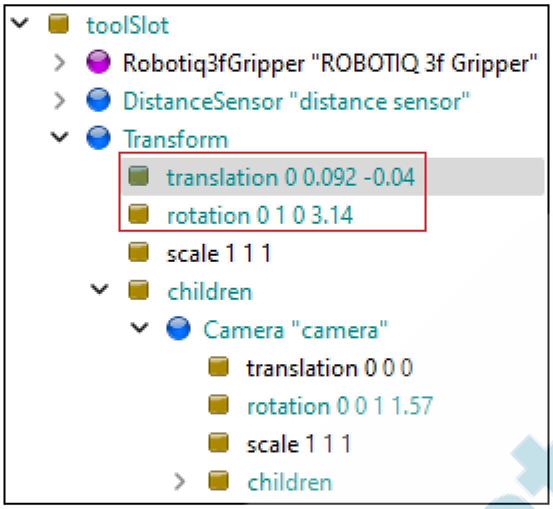
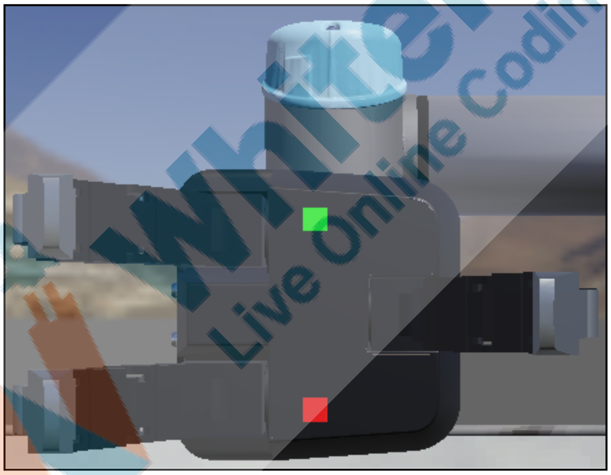
You will see that the **x axis** or the **red colored axis** is pointing **outwards** in the gif shown, but the **z axis** or the **blue colored axis** of the camera node is still in downward direction.

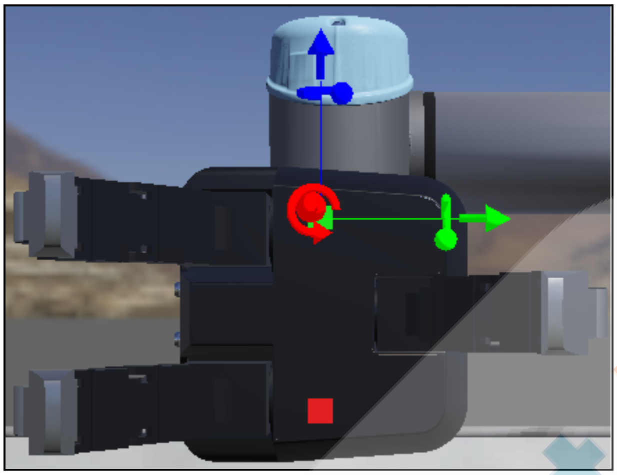
<table>
<tr><td colspan="2">
toolSlot
Robotiq3fGripper "ROBOTIQ 3f Gripper"
DistanceSensor "distance sensor"
Transform
translation 0 0.092 -0.04
rotation 0 1 0 0
scale 1 1 1
children
Camera "camera"
translationStep 0.01
rotationStep 0.262
staticBase TRUE</td></tr>
<tr><td>To make sure that the <b>z axis</b> of the <b>camera,</b> points in the <b>upward</b> direction, let’s rotate the <b>transform</b> node in <b>y</b> direction by <b>3.14 radians</b> or <b>180 degrees</b> by changing its <b>rotation</b> property to <b>0</b> in <b>x direction, 1</b> in <b>y direction, 0</b> in <b>z direction</b> and specify the <b>rotation angle</b> as <b>3.14 radians</b>.

After changing the orientation, you still won’t be able to see the camera. Can you tell me why?

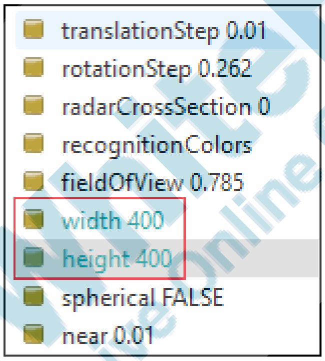
It is still inside the gripper hand and we need to change the translation property of the <b>transform</b> node.

Let’s change it to <b>0</b> in <b>x direction, 0.092</b> in <b>y direction</b> and <b>-0.04</b> in the <b>z direction</b>.</td><td>ESR : Varied.</td></tr>
</table>

 toolSlot Robotiq3fGripper "ROBOTIQ 3f Gripper" DistanceSensor "distance sensor" Transform translation 0 0.092 -0.04 rotation 0 1 0 3.14 scale 1 1 1 children Camera "camera" translation 0 0 0 rotation 0 0 1 1.57 scale 1 1 1 children	
Save your work till here and close all the nodes. Your camera is repositioned as shown in the graphic.	
	
Your camera is reoriented as shown in the graphic.	



Also, change the **resolution** of the camera to **400 px** in **width** and **400 px** in **height**.



translationStep 0.01
rotationStep 0.262
radarCrossSection 0
recognitionColors
fieldOfView 0.785
width 400
height 400
spherical FALSE
near 0.01

Teacher Stops Screen Share

So now it's your turn.
Please share your screen with me.

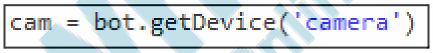
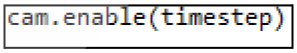
STUDENT-LED ACTIVITY 15 mins

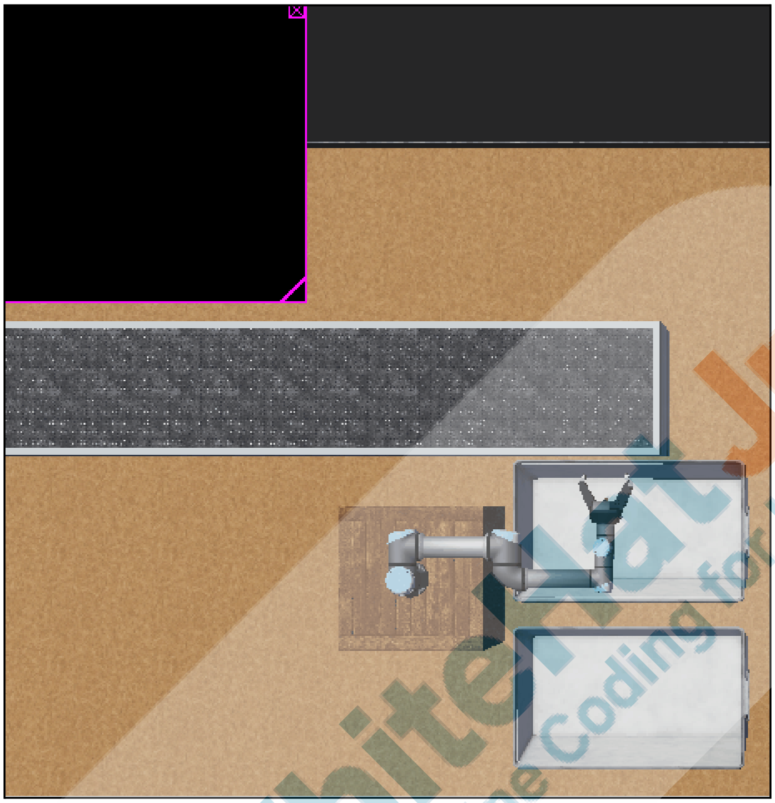
- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

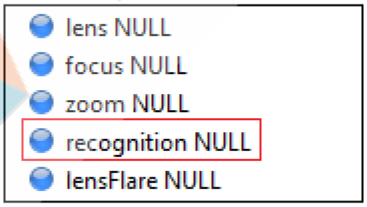
ACTIVITY

- **Getting video feed from the camera.**
- **Sorting objects using detection and recognition algorithms.**
- **Adding texture to the solid boxes.**

Teacher Action	Student Action
Finally, we are all set with the design part. Let's write some code so that we can fetch a live video feed from the camera and perform the detection and recognition algorithms. For that, first let's open the student boilerplate link and download all the files from it. Open all the downloaded files in the webots software. In the controller code, first let's create an **object** named **cam** for the camera node using the **.getDevice()** method of the **robot** class.	
 `cam = bot.getDevice('camera')`	
Next, let's enable the camera using the **.enable()** method. The **enable()** method takes the **controller timestep** as the argument.	
 `cam.enable(timestep)`	
Save your work till here and run the simulation. You will see a **live video** feed but your camera will not recognize anything yet.	

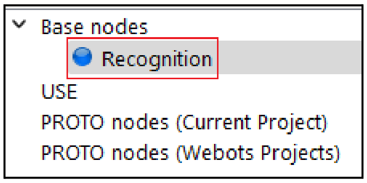


To make sure that your camera recognizes the solid box in front of it, scroll down into the **camera** properties and double click on the **recognition** property, which says **‘NULL’** for now.



- lens NULL
- focus NULL
- zoom NULL
- recognition NULL
- lensFlare NULL

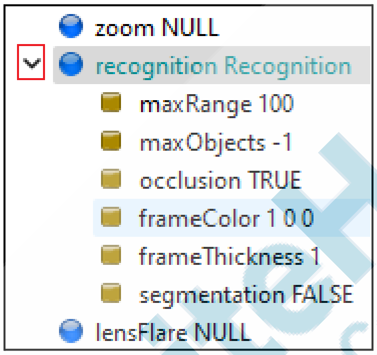
Expand the **base nodes** and double click on the **Recognition** node.



Base nodes
Recognition
USE
PROTO nodes (Current Project)
PROTO nodes (Webots Projects)

You will see that a **Recognition node** is added in the scene tree, under the **recognition property** of the camera.

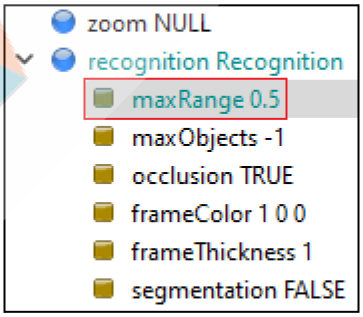
Expand it.



zoom NULL
recognition Recognition
maxRange 100
maxObjects -1
occlusion TRUE
frameColor 1 0 0
frameThickness 1
segmentation FALSE
lensFlare NULL

Change **max range** property to **0.5 m**, which means that your camera will only detect objects, if they are within a range of **0.5 m**.

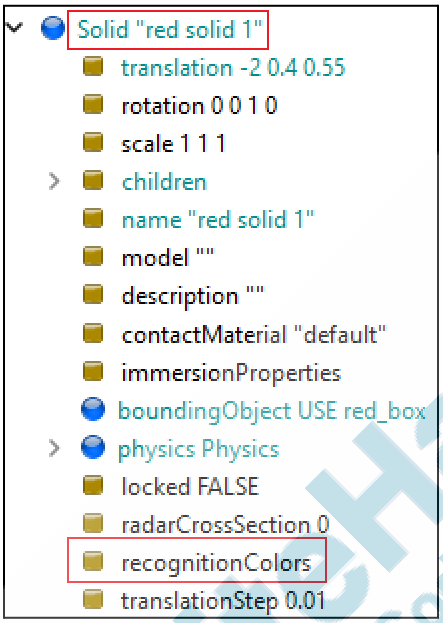
If you want to change other properties like **frame color** and **frame thickness**, you can, but they are fine for now.



zoom NULL
recognition Recognition
maxRange 0.5
maxObjects -1
occlusion TRUE
frameColor 1 0 0
frameThickness 1
segmentation FALSE

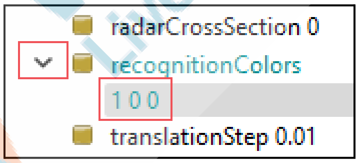
After adding the recognition node to your camera, there is one last thing you need to do to make the solid boxes recognized by the camera.

Expand the **red solid 1 node** and you will see the **recognition colors property**.



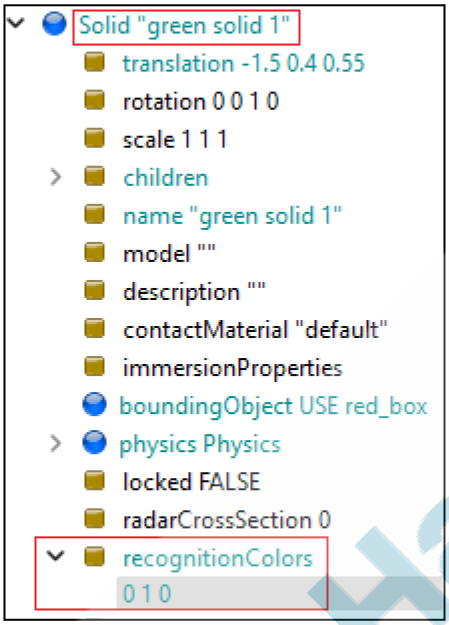
```
Solid "red solid 1"
  translation -2 0.4 0.55
  rotation 0 0 1 0
  scale 1 1 1
  children
  name "red solid 1"
  model ""
  description ""
  contactMaterial "default"
  immersionProperties
  boundingObject USE red_box
  physics Physics
  locked FALSE
  radarCrossSection 0
  recognitionColors
  translationStep 0.01
```

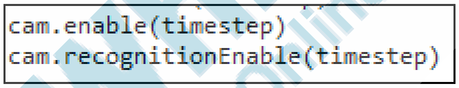
Double click on it and change the **recognition color** to **red**, for the **red solid 1 node**. You can do it by specifying the **red value** as **1**, **green value** to **0** and **blue value** to **0** as well.



```
radarCrossSection 0
recognitionColors
  1 0 0
translationStep 0.01
```

Do the same for **green solid 1** node and specify its **recognition colors** to **green**.

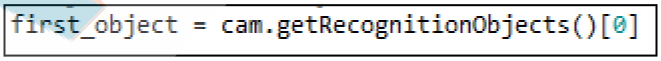
 Solid "green solid 1" translation -1.5 0.4 0.55 rotation 0 0 1 0 scale 1 1 1 children name "green solid 1" model "" description "" contactMaterial "default" immersionProperties boundingObject USE red_box physics Physics locked FALSE radarCrossSection 0 recognitionColors 0 1 0	
Now in our python code, let's enable the **recognition** property of our camera using the **.recognitionEnable() method**. It takes the controller **timestep** as the **argument**.	


`cam.enable(timestep)`
`cam.recognitionEnable(timestep)`	

Save your work till here and run the simulation. You will that your objects are being recognized now, but there is one more challenge? Can you tell me what it is? Great, that is correct, which means we still have to add some part in the code to make it work.	ESR : The camera is able to recognize the objects, but still it is not able to sort it.

After the recognition is working, let's get the data for the first object which is being recognized by the camera using the **.getRecognitionObjects()** method.

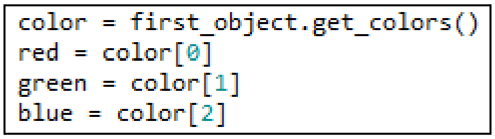
The above method returns a number of detected objects within a range of **0.5 m**. The **0th** index ensures that the data returned if for the **first object**.



```
first_object = cam.getRecognitionObjects()[0]
```

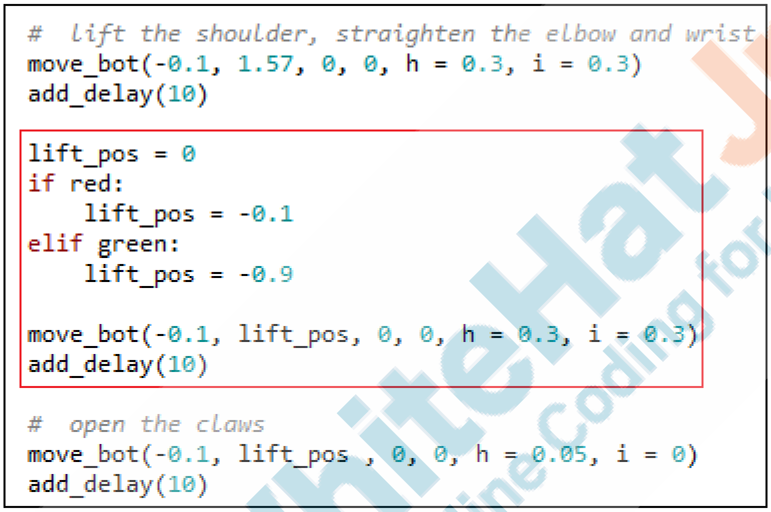
After you get the data for the first object, let's extract the color of the object from it using the **.get_colors()** method.

This method will return **RGB** content of a color in the form of a **list**. We can then parse the **red, green** and **blue** values of the detected color using the **list indexing**.



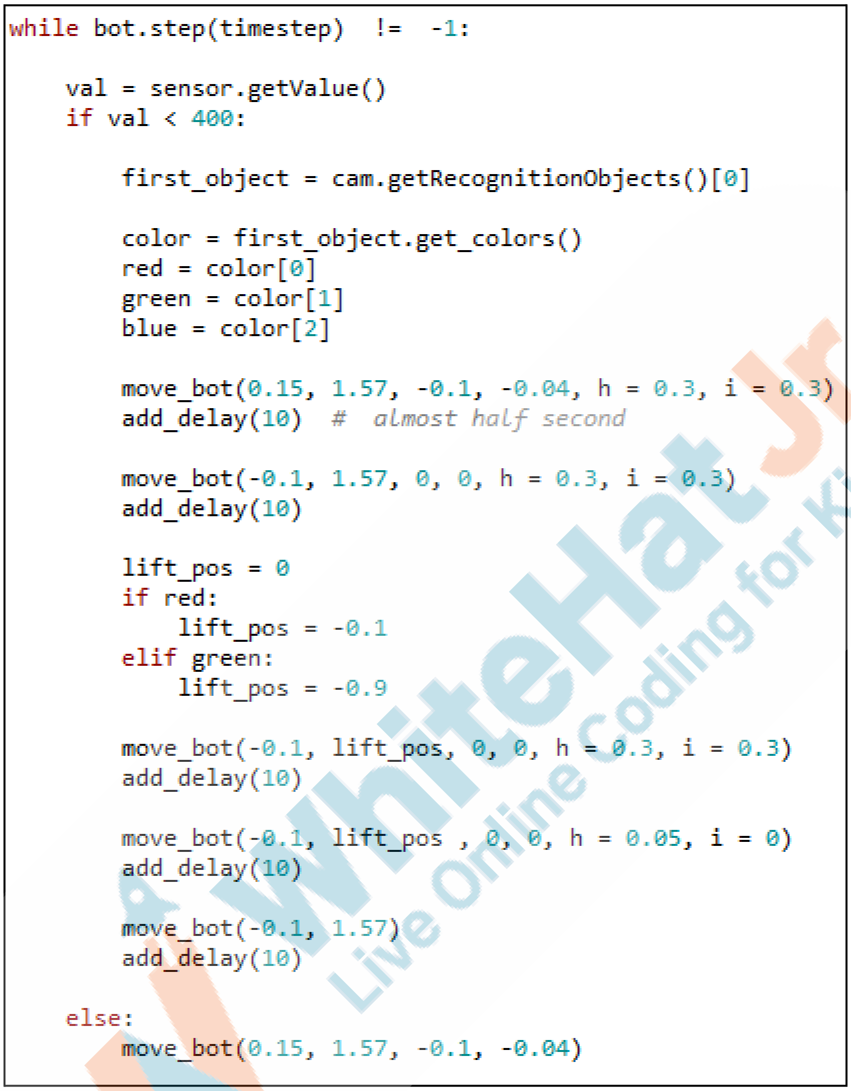
```
color = first_object.get_colors()
red = color[0]
green = color[1]
blue = color[2]
```

Finally, let’s add some conditionals, to move your arm over **‘red box 1’ if** the **red solid block** is detected, or **‘green box 1’ if** the **green solid block is** detected.



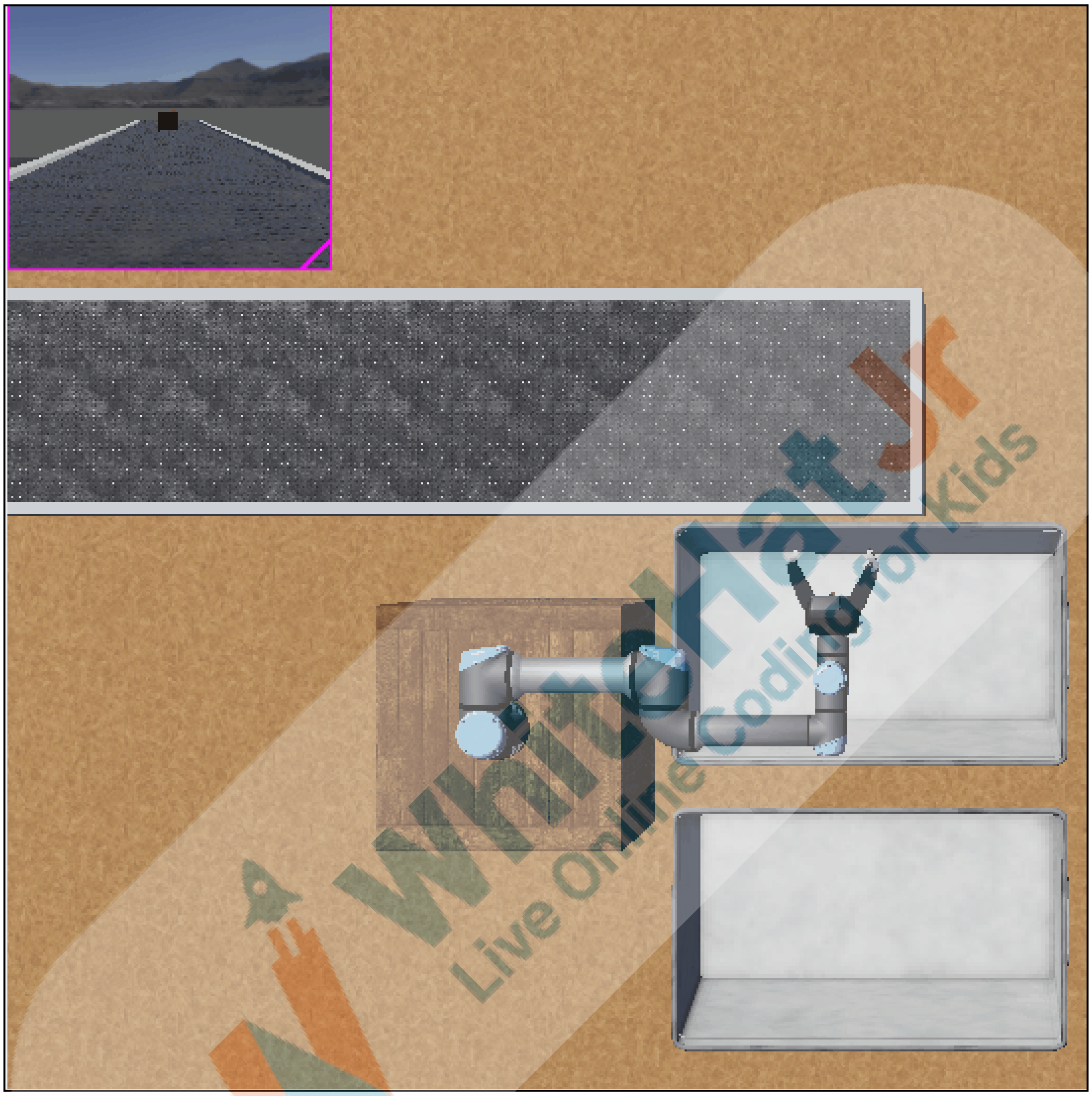
```
#  lift the shoulder, straighten the elbow and wrist
move_bot(-0.1, 1.57, 0, 0, h = 0.3, i = 0.3)
add_delay(10)

lift_pos = 0
if red:
    lift_pos = -0.1
elif green:
    lift_pos = -0.9

move_bot(-0.1, lift_pos, 0, 0, h = 0.3, i = 0.3)
add_delay(10)

#  open the claws
move_bot(-0.1, lift_pos , 0, 0, h = 0.05, i = 0)
add_delay(10)
```

The complete code will look as shown in the graphic.



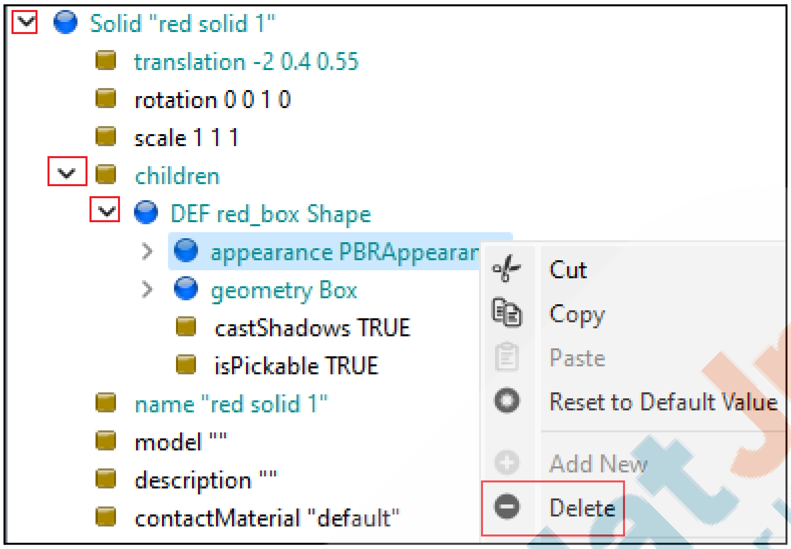
```
while bot.step(timestep)  !=  -1:

    val = sensor.getValue()
    if val < 400:

        first_object = cam.getRecognitionObjects()[0]

        color = first_object.get_colors()
        red = color[0]
        green = color[1]
        blue = color[2]

        move_bot(0.15, 1.57, -0.1, -0.04, h = 0.3, i = 0.3)
        add_delay(10)  #  almost half second

        move_bot(-0.1, 1.57, 0, 0, h = 0.3, i = 0.3)
        add_delay(10)

        lift_pos = 0
        if red:
            lift_pos = -0.1
        elif green:
            lift_pos = -0.9

        move_bot(-0.1, lift_pos, 0, 0, h = 0.3, i = 0.3)
        add_delay(10)

        move_bot(-0.1, lift_pos , 0, 0, h = 0.05, i = 0)
        add_delay(10)

        move_bot(-0.1, 1.57)
        add_delay(10)

    else:
        move_bot(0.15, 1.57, -0.1, -0.04)
```

Finally, let’s save our work and run the simulation.



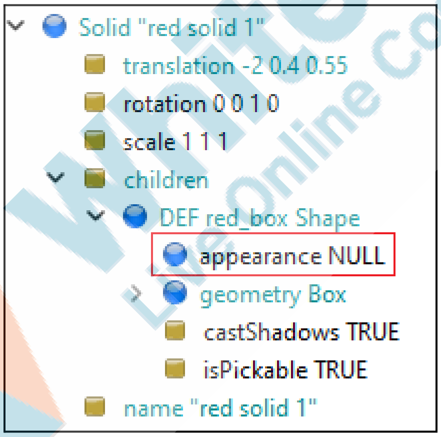
Click here to view the reference video.

Great, your project is working fine. Let's do some final touch ups and add a **rusty metal** or **galvanized metal textures** over the solid blocks. For that, expand the **red solid 1** node, and **delete** its **appearance** as shown in the graphic.	



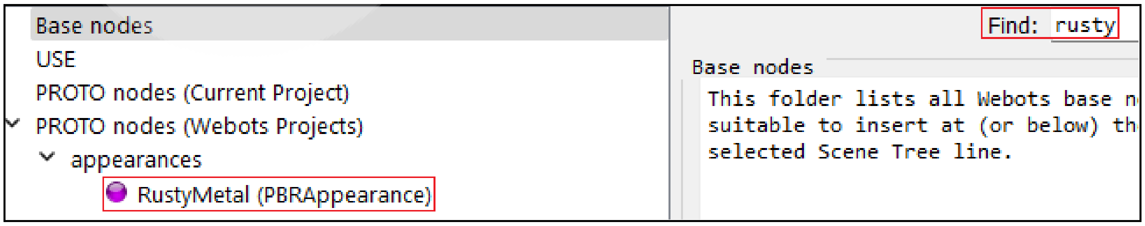
```
Solid "red solid 1"
    translation -2 0.4 0.55
    rotation 0 0 1 0
    scale 1 1 1
    children
      DEF red_box Shape
        appearance PBRAppearan
        geometry Box
        castShadows TRUE
        isPickable TRUE
    name "red solid 1"
    model ""
    description ""
    contactMaterial "default"
```

Once the **appearance property** is **NULL**, double click on it.



```
Solid "red solid 1"
    translation -2 0.4 0.55
    rotation 0 0 1 0
    scale 1 1 1
    children
      DEF red_box Shape
        appearance NULL
        geometry Box
        castShadows TRUE
        isPickable TRUE
    name "red solid 1"
```

Search the word **'rusty'** in the **Find textbox** and double click on the **Rusty metal appearance (PBR Appearance)**.

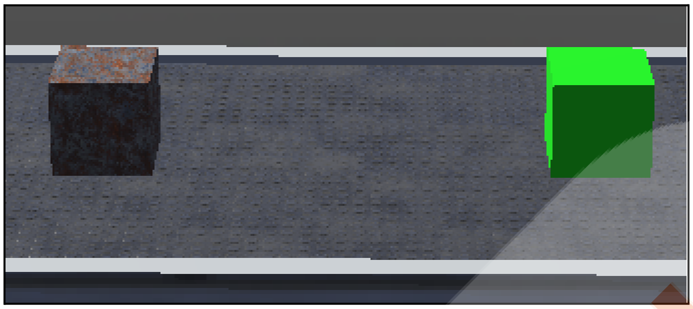
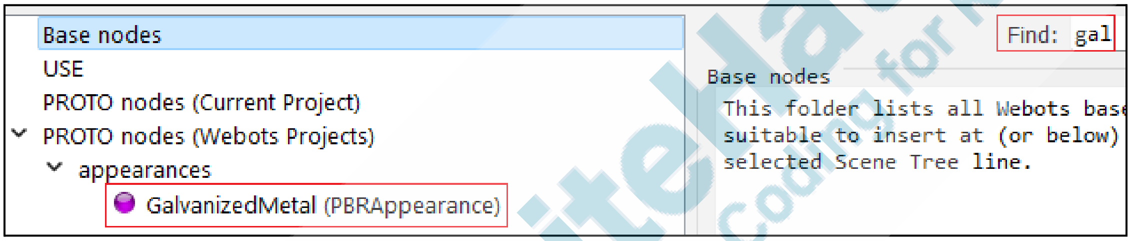
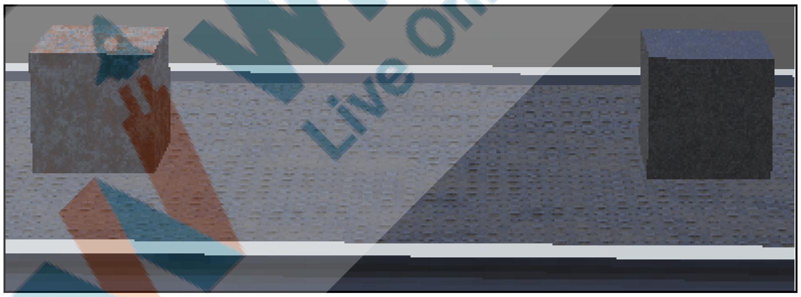


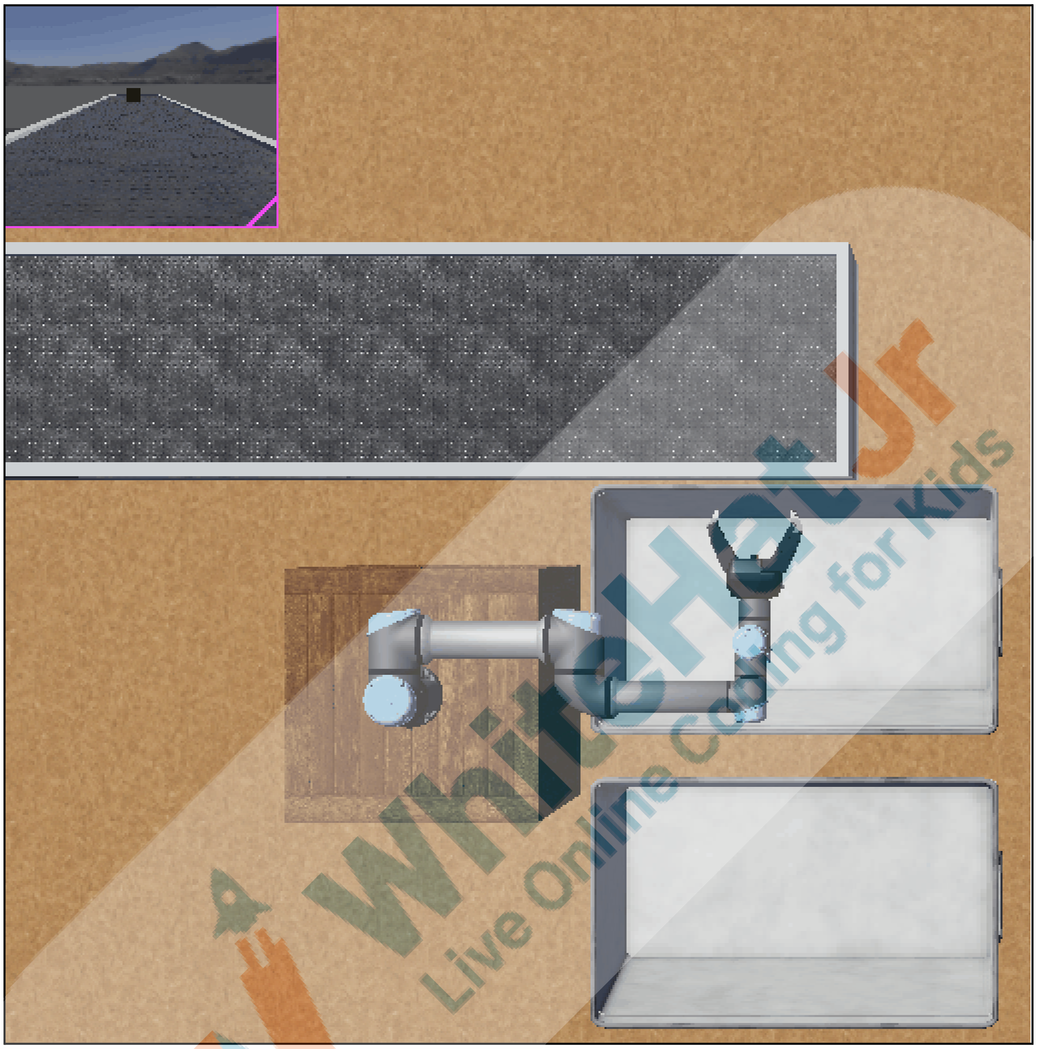
```
Base nodes
USE
PROTO nodes (Current Project)
PROTO nodes (Webots Projects)
  appearances
    RustyMetal (PBRAppearance)
```

Find: rusty

Base nodes

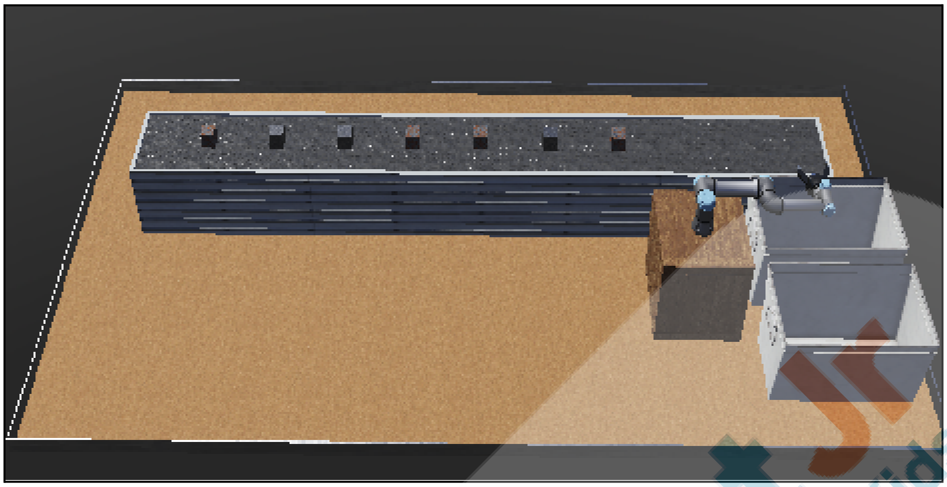
This folder lists all Webots base n
suitable to insert at (or below) th
selected Scene Tree line.

Your red solid block will get a texture of rusty metal.	
	
Do the same thing for the **green solid 1** and give it an appearance of **galvanized metal (PBR Appearance)**	
 Base nodes USE PROTO nodes (Current Project) PROTO nodes (Webots Projects) appearances GalvanizedMetal (PBRAppearance) Find: gal Base nodes This folder lists all Webots base suitable to insert at (or below) selected Scene Tree line.	
Finally, your environment will look as shown in the graphic.	
	
Save your work and run the simulation.	

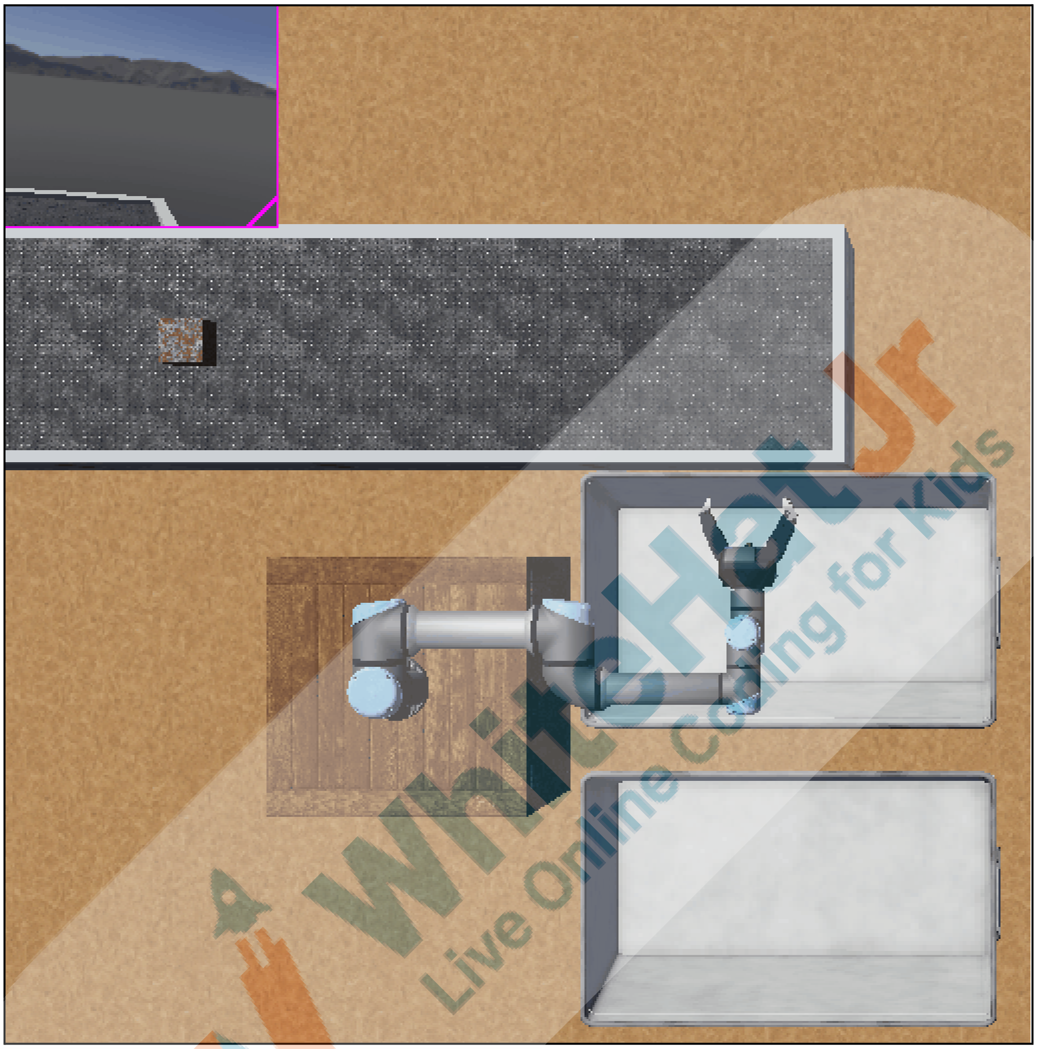


Click here to view the reference video.

Great, everything is working fine. Good job! You can copy and paste the solid blocks and make your factory environment look as shown in the graphic below.	



The final output would look as shown in the gif.	



Click here to view the reference video.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 mins

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.

- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	
You get “hats-off” for your excellent work! In the next class, we are going to start building a four legged robot.	

PROJECT OVERVIEW DISCUSSION
Refer the document below in Activity Links Sections

Teacher Clicks ✖ End Class

ACTIVITY LINKS

Activity Name	Description	Links

Teacher Activity 1	Teacher boilerplate	https://github.com/procodingclass/PRO-C291-Teacher-Boilerplate.git
Teacher Reference 1	Reference Code	https://github.com/procodingclass/PRO-c291-Reference-Code.git
Teacher Reference 2	Project	https://s3-whjr-curriculum-uploads.whjr.online/1e332817-4b23-422a-a7d1-27c0339de643.pdf
Teacher Reference 3	Project Solution	https://github.com/procodingclass/PRO-C291-Project-Solution
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/1a34e7b0-9c99-4348-8a45-e31dbca82637.pdf
Teacher Reference 5	Final output gif	https://s3-whjr-curriculum-uploads.whjr.online/d9215a12-0226-4151-8f21-4f4f9d318535.mp4
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C291-Student-Boilerplate.git