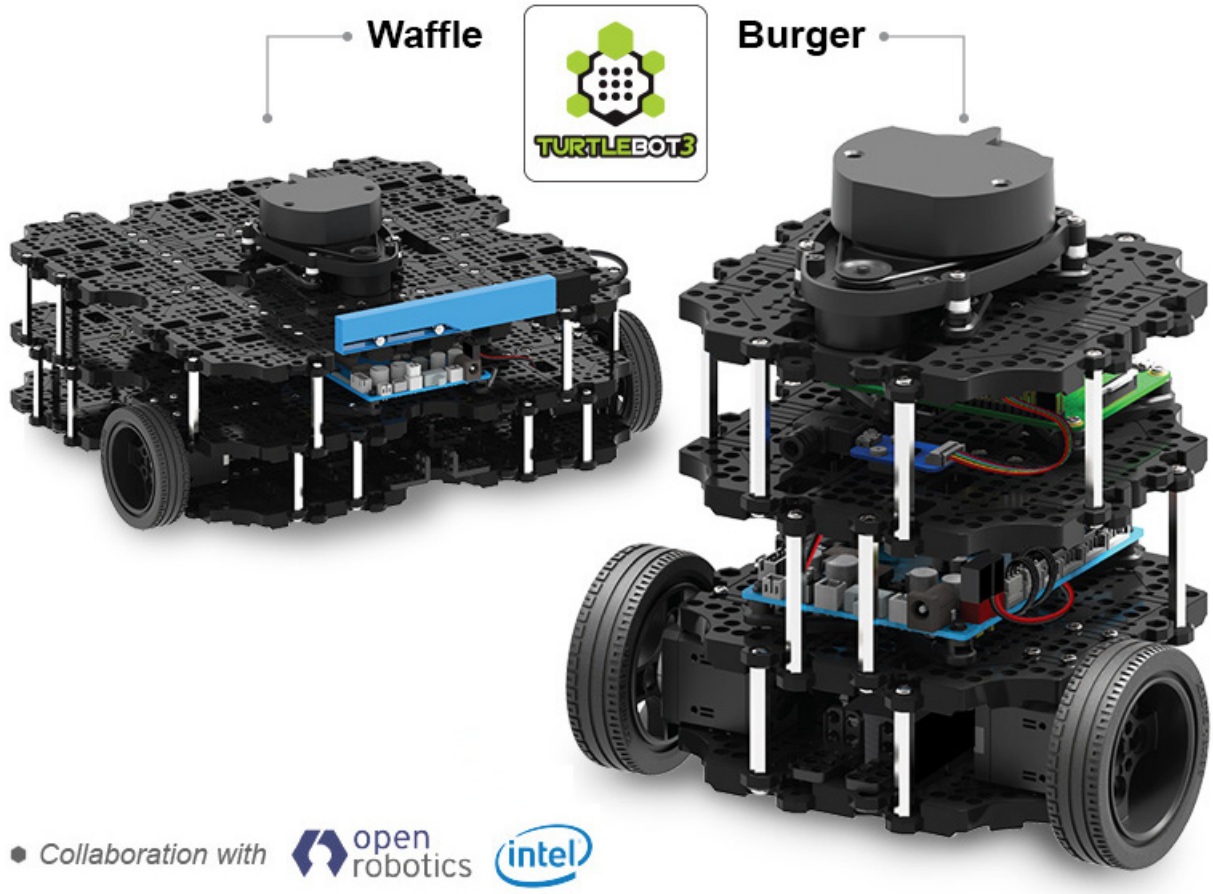
TurtleBot3Blockly Documentation

Release 0.0.1

Aravind Krishnan

Oct 29, 2017

Contents:



A detailed documentation on how to use Blockly (free and open source software) with TurtleBot3.

CHAPTER 1

About

Dabit Industries is an official reseller of the latest ROS based development platform - TurtleBot3. We want to help our customers to quickly understand how to get started programming the TurtleBot3. The usual requirements are to have a background in Linux, software engineering and some bit of robotics. Now, we want to introduce **Blockly** and make the TurtleBot3 much easier to program while having fun.

Blockly is a free and open source web interface that can be used to program the TurtleBots. Its intuitive drag-and-drop based programming style welcomes everyone to try out the logic based programming and is easy to get started.

We will help you understand the required concepts necessary to work with Blockly and get your TurtleBot up and running in no time!

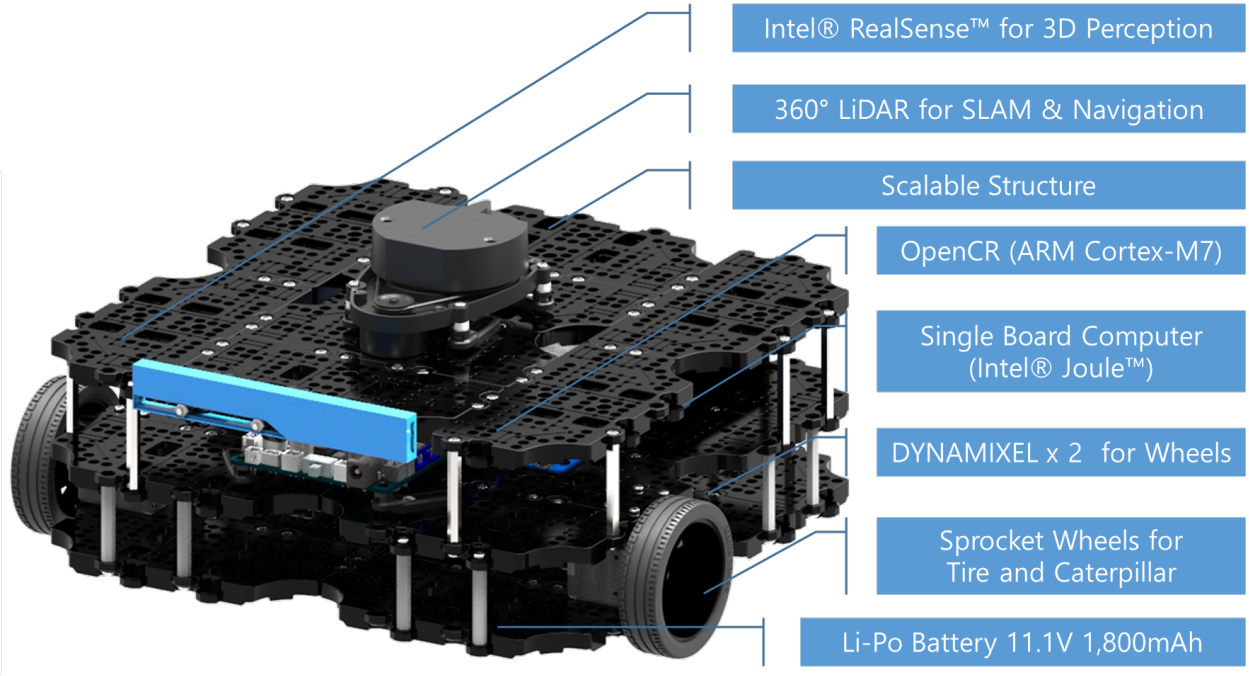
You can order the TurtleBot3 here.

TurtleBot3 is available in two variants:

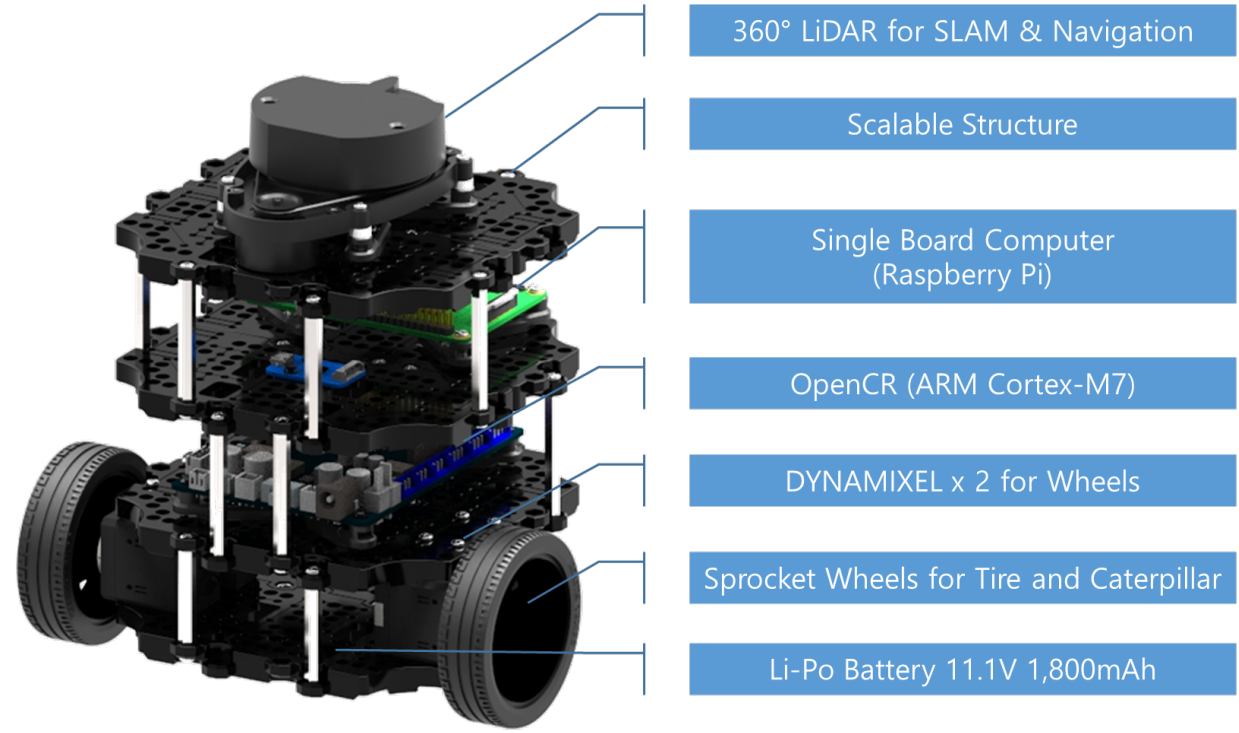
- Waffle
- Burger

Waffle comes with an Intel ® RealSense ™ camera for 3D perception and the Burger doesn't. All other sensors remain the same in both models. The Burger is a stacked up model of the TurtleBot3 whereas Waffle has a wider base (perhaps intentionally named after the way they look).

1.1 Waffle



1.2 Burger



CHAPTER 2

Software Setup

To program a TurtleBot3 with Blockly some software packages must be installed. This section on software setup is also helpful for those who would like to create custom blocks to either add new funtionalities or modify the existing ones.

2.1 TurtleBot3 + Remote PC + Blockly

Before we begin the software setup let's understand, on a high level, how the three things are connected.Also note that TurtleBot3 and Remote PC should be connected to the same WiFi network.

Remote PC

A Desktop PC or Laptop a.k.a Remote PC should have `Ubuntu 16.04` and `ROS Kinetic Kame` installed. Remote PC will run ROS and be the `ROS_MASTER`.

TurtleBot3

TurtleBot3 runs a custom Linux version called `Ubuntu Mate` and will need its own setup. Once TurtleBot3 is up and running ROS, it should connect to the Remote PC and recognize it as the `ROS_MASTER`.

Blockly

Now that the Remote PC and TurtleBot3 are connected, you need to setup Blockly software package and launch it. This way, using ROS, Blockly can send commands to TurtleBot3.

The steps below will help you install and launch Blockly on the Remote PC. They are adapted from the instructions provided by Erle Robotics.

2.2 Installation

2.2.1 Linux

Note: TurtleBot3s are tested on `Ubuntu 16.04` and `ROS Kinetic Kame`. So, these are two prerequisites to setup Blockly and work with a TurtleBot3.

Open terminal and enter the following instructions to install and develop Blockly.

```
$ mkdir -p ~/blockly_ws/src
$ cd ~/blockly_ws/src
$ git clone https://github.com/dabit-industries/turtlebot3_blockly
$ cd turtlebot3_blockly/frontend/
$ git submodule add https://github.com/dabit-industries/ace-builds.git ace-builds
$ git submodule init
$ git submodule update
$ git submodule add https://github.com/dabit-industries/blockly.git blockly
$ git submodule init
$ git submodule update
$ cd ~/blockly_ws/
$ catkin_make_isolated -j2 --pkg turtlebot3_blockly --install
```

or

```
$ mkdir -p ~/blockly_ws/src
$ cd ~/blockly_ws/src
$ git clone --recurse-submodules https://github.com/dabit-industries/turtlebot3_
↪blockly
$ cd ~/blockly_ws
$ catkin_make_isolated -j2 --pkg turtlebot3_blockly --install
```

CHAPTER 3

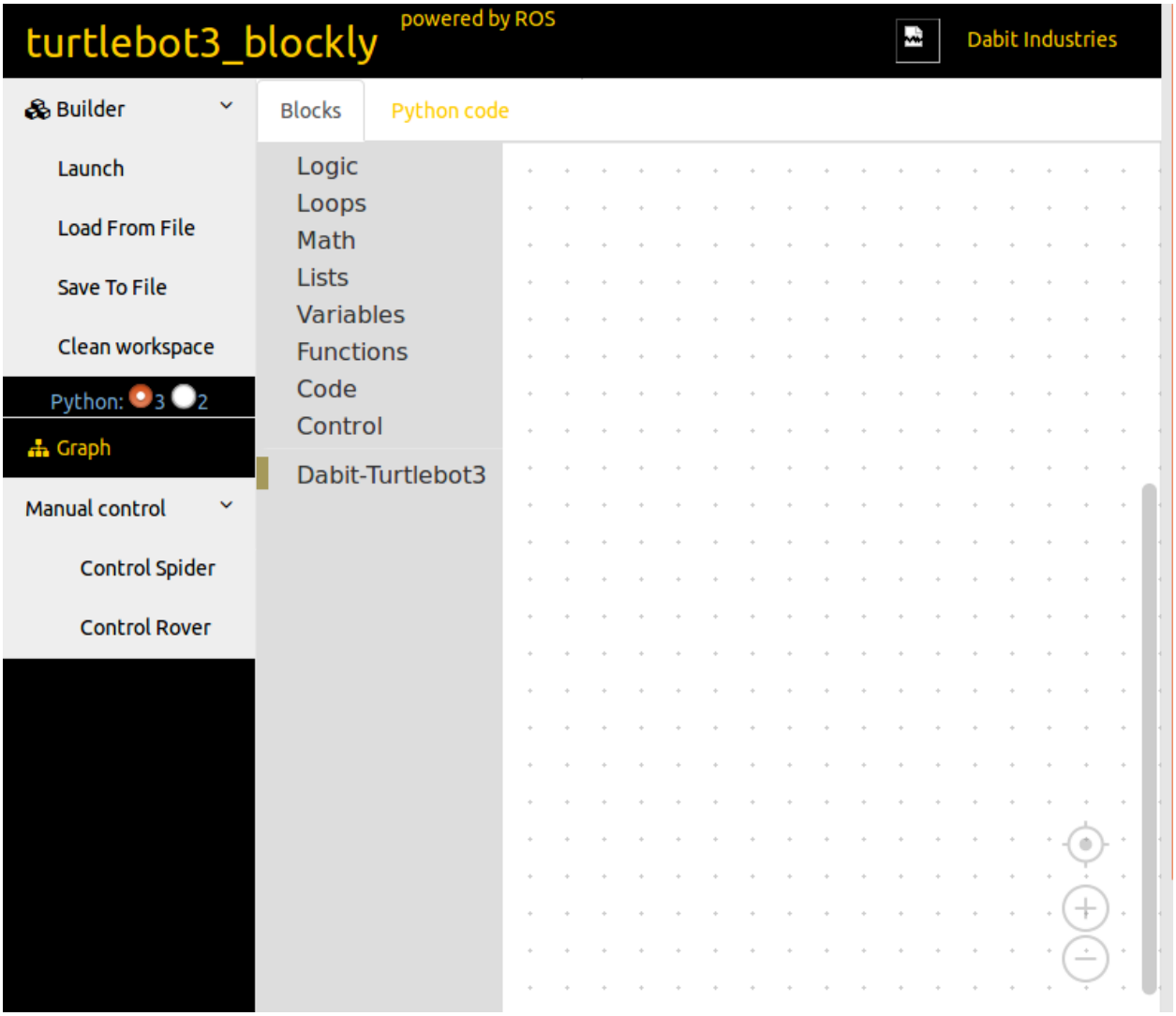
Launch Blockly web interface

Once the software package is setup there are a few commands to launch the Blockly web interface. Open terminal and type the following commands and leave the terminal running.

```
$ cd ~/blockly_ws
$ source devel_isolated/setup.bash
$ roslaunch turtlebot3_blockly turtlebot3_blockly.launch
```

Note: To launch the web interface of blockly, you don't necessarily have to start `roscore` but you should if you plan to connect TurtleBot3 and test it during development.

Open a web browser and type `127.0.0.1:1036` in the address bar. The Blockly web interface should open and will look like the image below.


turtlebot3_blockly
powered by ROS
Dabit Industries
Builder
Launch
Load From File
Save To File
Clean workspace
Python: 3 2
Graph
Manual control
Control Spider
Control Rover
Blocks
Python code
Logic
Loops
Math
Lists
Variables
Functions
Code
Control
Dabit-Turtlebot3

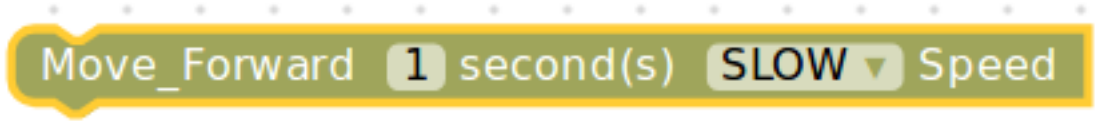
CHAPTER 4

Basic Maneuvers

Are you ready to make the TurtleBot3 perform basic actions like moving forward, backward, turning right, left and stop?

It's quite simple in Blockly. You just need to select the appropriate block from the list of blocks available. Let's look at each of the basic actions and program the TurtleBot3 to move accordingly.

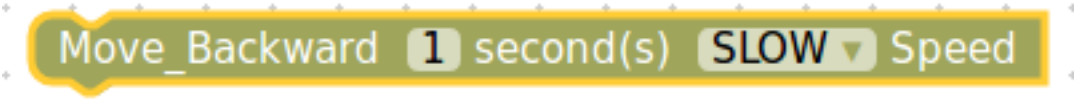
4.1 Moving Forward



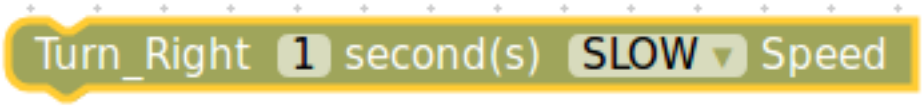
This block makes the TurtleBot3 move forward in three preset speed values - **SLOW**, **NORMAL** and **FAST** for a desired amount of time in seconds. Behind this block there is a short piece of code in Python that talks to the robot from the Blockly web interface and commands it to move forward with one of the preset values as programmed by the user. The figure below shows how the block is dragged into the workspace. The time and speed values are then changed.

The other basic maneuver blocks of the TurtleBot3 are shown in the following figures.

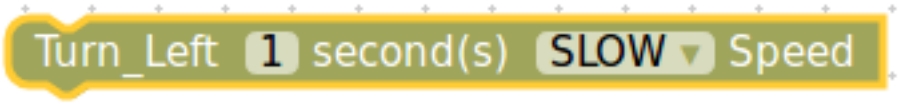
4.2 Moving Backward



4.3 Turning Right

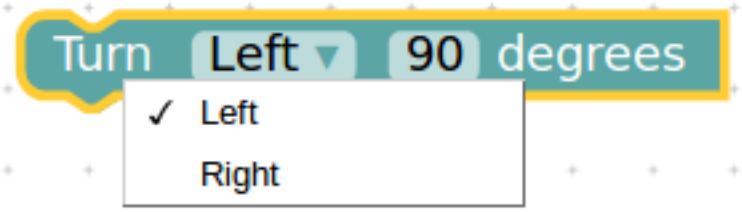


4.4 Turning Left



4.5 Turn Left/Right in degrees

All the commands shown above move the TurtleBot3 for a desired amount of time. To turn the TurtleBot3 a specific amount (degree) you can use the **Turn Left/Right __ degrees** command. Left turn is counter-clockwise and right turn is clockwise direction. This block uses the data from an Inertial Measurement Unit (IMU) sensor.



4.6 Stop

You will find out how to load the program onto the TurtleBot3 and make it move, in the next page.

CHAPTER 5

A Simple Program

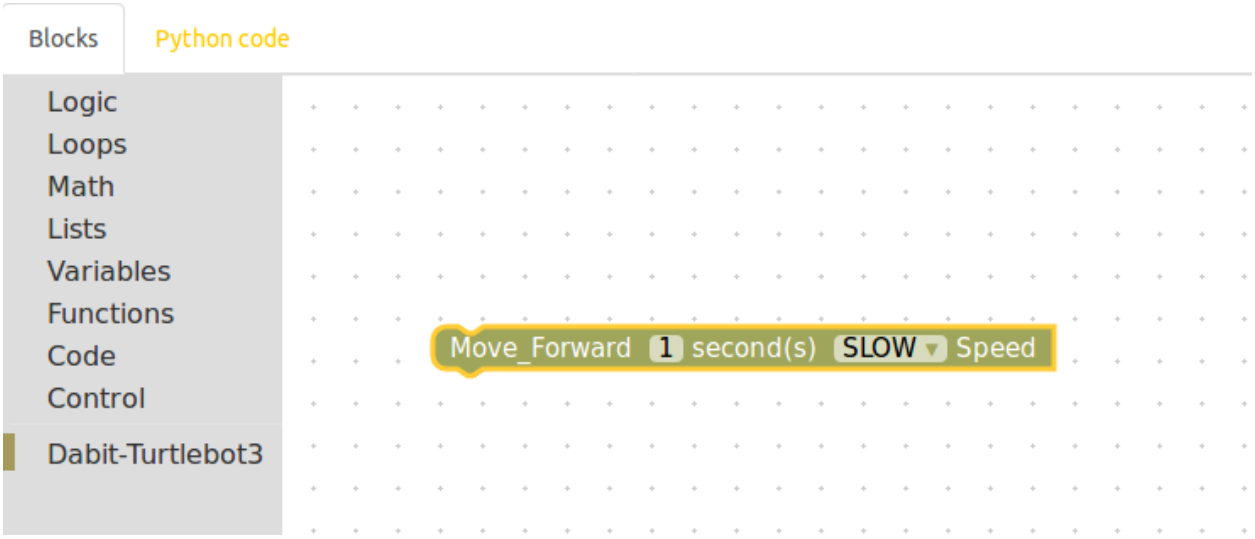
Let's write a simple program to make the TurtleBot3 do the following tasks consecutively:

1. Move forward for two seconds
2. Wait for one second
3. Move backward for three seconds
4. Wait for one second

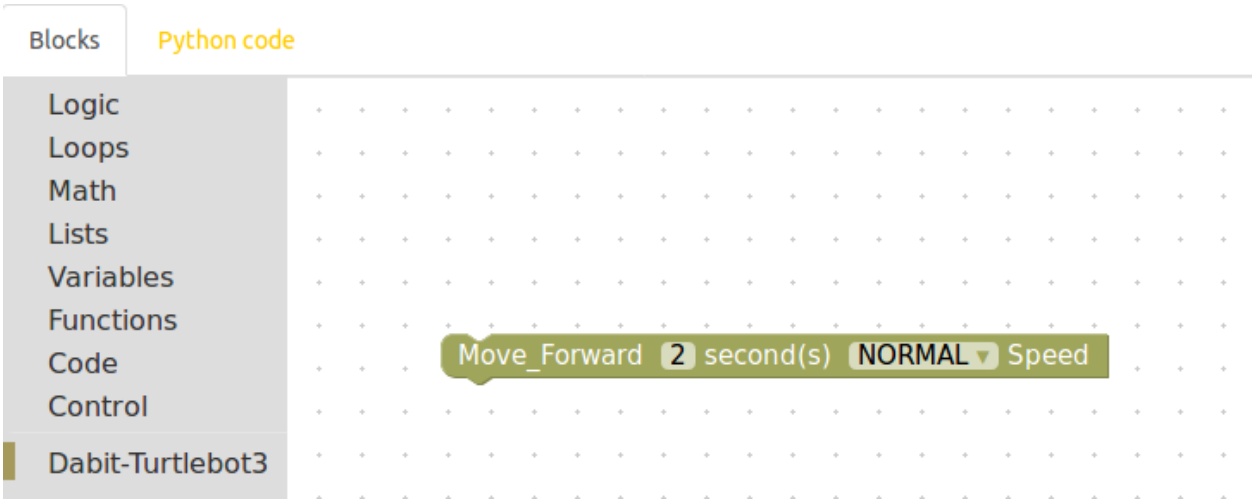
5.1 Drag and drop blocks

Launch the Blockly web interface and ensure that TurtleBot3 is connected to the master (your pc/laptop).

On the left sidebar of the Blockly web interface you will find `Dabit-Turtlebot3` icon. Click on it and drag a `Move_Forward` block onto the workspace.



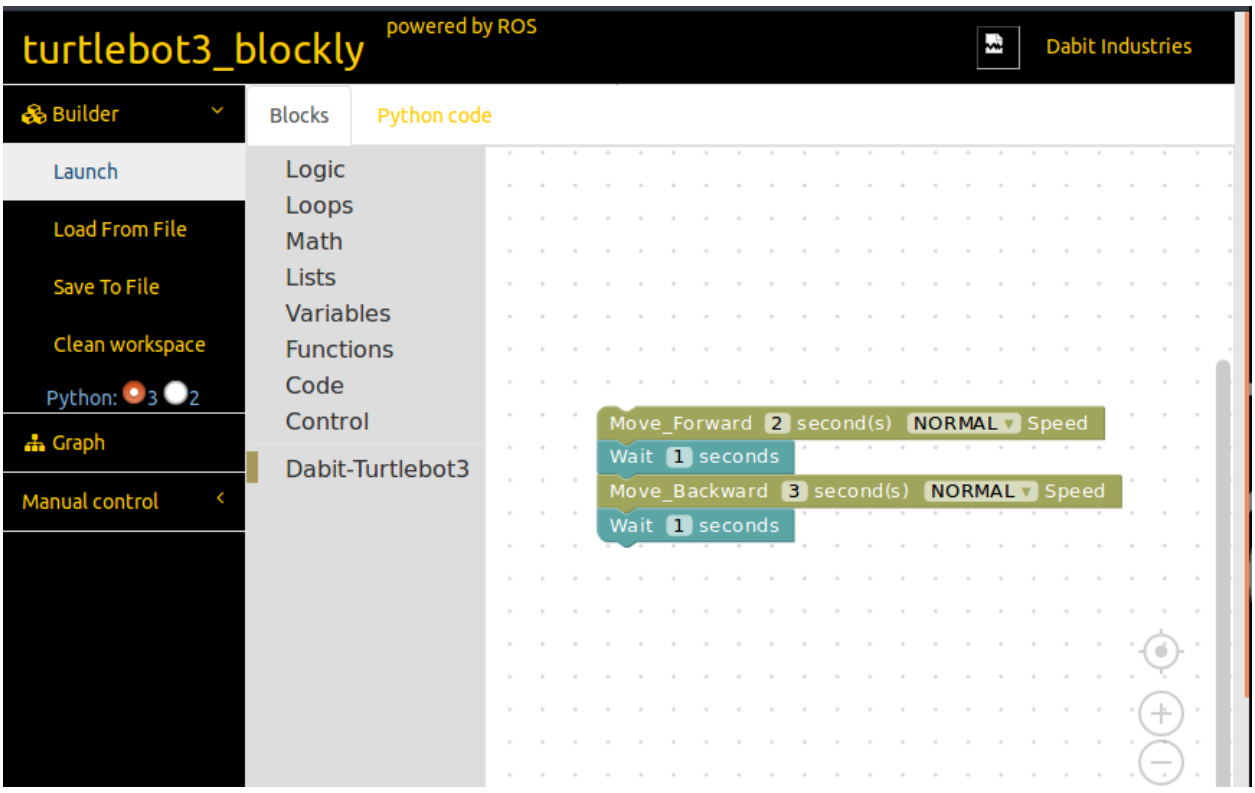
Our task is to move the TurtleBot3 for two seconds, so change the seconds field to `2` and let's run it at a **NORMAL** speed.



The following figure shows the remaining steps to complete the program. You can find the `Wait 1 seconds` block inside the `Control` icon on the left sidebar.

5.2 Launch the program

After writing the program you should launch it to make the TurtleBot3 move. Click once on the `Launch` icon on the far left of the Blockly web interface and you should see TurtleBot3 perform the tasks consecutively.



CHAPTER 6

Block Creation - Overview

By now you have an idea that `blocks` are the fundamental elements in this drag and drop Blockly sofware. We will discuss how to create new blocks and/or to edit the existing ones.

This guide is adapted from Erle Robotics documentation on Block Creation and we will focus on the block creation pertaining to TurtleBot3's functionalities.

6.1 Understanding the file structure

Code changes or development typically happens inside the **blockly_ws/src/** folder.

There are four different files one should edit to create a new block.

```
- turtlebot3_blockly/frontend/blockly/generators/python/scripts/turtlebot3/example.py
- turtlebot3_blockly/frontend/blockly/generators/python/customName.js
- turtlebot3_blockly/frontend/blockly/blocks/customName.js
- turtlebot3_blockly/frontend/pages/blockly.html
```

A detailed description of the contents of these four files, in a particular order will help in creating or editing a block.

Note: The filenames with the `.js` extension must be the same.

Let's take a look at the **blockly/** directory

```
$ cd turtlebot3_blockly/frontend/blockly/
```

Note: `blockly` was one of the submodules that we cloned during the software setup.

The directories **blocks/** and **generators/** contain a few files that we must edit. Each block that you see on the Blockly web interface has its own python script that provides the block's functionality.

6.1.1 The Python script

Let's look into the **generators/python/scripts/turtlebot3/** directory.

```
$ cd generators/python/scripts/turtlebot3/
```

This directory should contain a few python scripts already.

Move Forward is a custom block that moves the TurtleBot3 forward in one of three speed modes - SLOW, NORMAL and FAST. A python script called `move_forward.py` shown below is the backend code for this block.

```
import rospy, sys
import time
from geometry_msgs.msg import Twist

pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
#rospy.init_node('circle_mode', anonymous=True)
rate = rospy.Rate(10) # 10Hz
twist = Twist()
start = time.time()
flag=True #time flag
# Angular velocity = linear velocity / radius
speed=dropdown_speed # SLOW, NORMAL, FAST
twist.linear.z = 0.00

# CLOCKWISE rotation
if speed =='SLOW':
    twist.linear.y = 0.05
    twist.linear.x = 0.05
elif speed =='NORMAL':
    twist.linear.y = 0.25
    twist.linear.x = 0.25
elif speed == 'FAST':
    twist.linear.y = 0.75
    twist.linear.x = 0.75
while not rospy.is_shutdown() and flag:
    sample_time=time.time()
    if ((sample_time - start) > 3):
      flag=False
    pub.publish(twist)
twist = Twist()
pub.publish(twist)
rate.sleep()
```

6.1.2 The Javascript (Block functionality)

The Blockly web interface needs a javascript file that can link the python script of our custom block and to describe the fields of the block. For instance, SLOW, NORMAL and FAST are one of the fields of a block.

```
$ cd ~/turtlebot3_blockly/frontend/blockly/generators/python/dabit-turtlebot3.js
```

Once the `dabit-turtlebot3.js` file opens, look for the particular code section written to link the **move_forward** block. It should look something similar to this image below.

```
27
28  Blockly.Python['move_forward'] = function(block) {
29
30    var dropdown_speed = block.getFieldValue('speed');
31
32    var code = "";
33    code += "dropdown_speed = \"" + dropdown_speed.toString() + "\"\n";
34    code += Blockly.readPythonFile("../blockly/generators/python/scripts/turtlebot3/move_forward.py");
35    return code;
36
37  };
38
```

6.1.3 The Javascript (Look and feel of the block)

In addition to the previous `dabit-turtlebot3.js` file there is one more with the same name inside the **blocks/** directory. Here we describe the look and feel of the block along with few other features - for instance, whether the block connects to any previous or future block(s).

```
$ cd ~/turtlebot3_blockly/frontend/blockly/blocks/dabit-turtlebot3.js
```

```
37
38  Blockly.Blocks['move_forward'] = {
39    init: function() {
40      this.appendDummyInput()
41          .appendField("Move_Forward ")
42          .appendField(new Blockly.FieldDropdown([["SLOW", "SLOW"], ["NORMAL", "NORMAL"], ["FAST", "FAST"]]), "speed")
43          .appendField("Speed");
44      this.setPreviousStatement(true);
45      this.setNextStatement(true);
46      this.setColour(65);
47      this.setTooltip('');
48      this.setHelpUrl('http://erlerobotics.com/docs/Robot_Operating_System/ROS/Blockly/Intro.html');
49    }
50  };
```

6.1.4 The HTML

Update the `blockly.html` file to reflect the changes of our custom block in the Blockly web interface.

The `<category>` tag contains the block details and below is an image that shows the contents of it.

```
384         <category id="turtlebot3" name="Dabit-Turtlebot3" colour="50">
385                 <block type="circle_mode"></block>
386                 <block type="move_forward"></block>
387                 <block type="move_backward"></block>
388         </category>
389
```

Now that you have an idea of what files to edit, let's look at the specifics of a block creation in the next page.

CHAPTER 7

Specifics of Block Creation

Blocks are of three types:

1. Block with an input
2. Block with an output
3. Block without an input or output

We looked at the `.js` files in the previous section. We should edit them to categorize these blocks into the three types as mentioned as above. Let's go over these types in detail.

7.1 Block with an input

To create a block with an input the two `.js` files to be edited are:

- turtlebot3_blockly/frontend/blockly/generators/python/customName.js
- turtlebot3_blockly/frontend/blockly/blocks/customName.js

We will be looking at the same example of **move_forward** block. The file inside ../generators/python/customName.js has the following code.

```
Blockly.Python['move_forward'] = function(block) {

var dropdown_speed = block.getFieldValue('speed');

var code = "";
code += "dropdown_speed = \"" + dropdown_speed.toString() + "\"\n";
code += Blockly.readPythonFile("../blockly/generators/python/scripts/turtlebot3/move_
↪forward.py");
return code;

};
```

The code snippet above takes in a value for speed {SLOW, NORMAL or FAST} from the user.

```
var dropdown_speed = block.getFieldValue('speed');
```

The `.js` file in the location ../blockly/blocks/customName.js has the following code

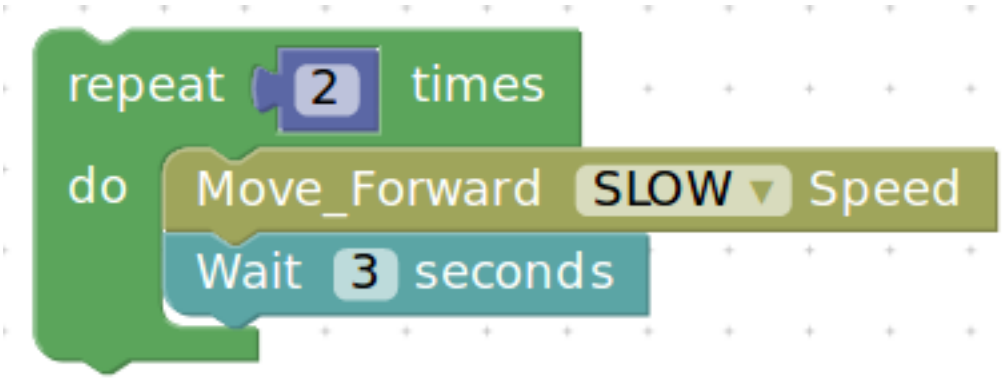
```
Blockly.Blocks['move_forward'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Move_Forward ")
        .appendField(new Blockly.FieldDropdown([["SLOW", "SLOW"], ["NORMAL", "NORMAL
↪"], ["FAST", "FAST"]]), "speed")
        .appendField("Speed");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setColour(65);
    this.setTooltip('');
    this.setHelpUrl('http://erlerobotics.com/docs/Robot_Operating_System/ROS/Blockly/
↪Intro.html');
  }
};
```

Apart from the input of the speed value from the user, we also see the following:

```
this.setPreviousStatement(true);
this.setNextStatement(true);
```

These two lines give the block a way to conenct to previous and future blocks.

For instance



The **repeat** block is connected to the **move_forward** block which is again connected to the **Wait** block.

The python script for the group of blocks is

```
for count in range(2):
  dropdown_speed = "SLOW"
  import rospy, sys
  import time
```

```
  from geometry_msgs.msg import Twist

  pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)
  #rospy.init_node('circle_mode', anonymous=True)
  rate = rospy.Rate(10) # 10Hz
  twist = Twist()
  start = time.time()
  flag=True #time flag
  # Angular velocity = linear velocity / radius
  speed=dropdown_speed # SLOW, NORMAL, FAST
  twist.linear.z = 0.00

  # CLOCKWISE rotation
  if speed =='SLOW':
      twist.linear.y = 0.05
      twist.linear.x = 0.05
  elif speed =='NORMAL':
      twist.linear.y = 0.25
      twist.linear.x = 0.25
  elif speed == 'FAST':
      twist.linear.y = 0.75
      twist.linear.x = 0.75
  while not rospy.is_shutdown() and flag:
      sample_time=time.time()
      if ((sample_time - start) > 3):
        flag=False
      pub.publish(twist)
  twist = Twist()
  pub.publish(twist)
  rate.sleep()
  import time
  time.sleep(3)
```

7.2 Block with an output

A block that does not take any input from the user but will have an output. A generic example of a clicking a picture with a camera module fixed to the turtlebot3.

The first `.js` file's code would look like

```
Blockly.Python['take_a_picture'] = function(block) {

window.open(
    '/pages/images/imageViewer.html',
    '_blank' // <- This is what makes it open in a new window.
);

var code = "";
code += Blockly.readPythonFile("../blockly/generators/python/scripts/brain/take_a_
↪picture.py");
return code;

};
```

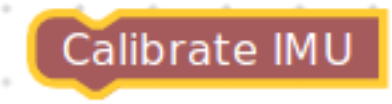
And the other `.js` file would have the following code

```
Blockly.Blocks['take_a_picture'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Take a picture");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setColour(0);
    this.setTooltip('');
    this.setHelpUrl('http://erlerobotics.com/docs/Robot_Operating_System/ROS/Blockly/
↪Intro.html');
  }
};
```

7.3 Block without an input or output

A block that typically configures some backend functionality without the need for an input or produce any output to the screen or on the robot.

An example would be to calibrate the IMU (Inertial Measurement Unit)



which doesn't necessarily have to take an input or produce an output, but simply calibrate the IMU with values hardcoded during the configuration.

And the two `.js` files

```
Blockly.Python['calibrate_imu'] = function(block) {

var code = "";
code += Blockly.readPythonFile("../blockly/generators/python/scripts/brain/calibrate_
↪imu.py");
return code;

};
```

```
Blockly.Blocks['calibrate_imu'] = {
  init: function() {
    this.appendDummyInput()
        .appendField("Calibrate IMU");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setColour(0);
    this.setTooltip('');
    this.setHelpUrl('http://erlerobotics.com/docs/Robot_Operating_System/ROS/Blockly/
↪Intro.html');
```

```
  }
};
```

CHAPTER 8

License

CHAPTER 9

Frequently Asked Questions

9.1 1. Where can I get more information about the TurtleBot3?

You can find more detailed information about the TurtleBot3 in the following link: http://turtlebot3.robotis.com/en/latest/introduction.html

9.2 2. Where can I order the TurtleBot3 (Waffle or Burger) from?

You can order them at our online shopping website https://dabit.industries/collections/turtlebot-3

9.3 3. What changed in the documentation recently?

changelog.rst

CHAPTER 10

Contact

For specific enquiries contact aravind AT dabit.industries

CHAPTER 11

Changelog - [development]

- Edited documentation text
- Fixed: “WARNING: Title underline too short” in faq.rst
- Added changelog to contents.rst
- Included seconds in basic blocks
- Launch Blockly page created
- Images and GIFs changed to show time in basic blocks
- Copyrights corrected
- Included Waffle and Burger images with description
- Explained how to write a simple program to move TurtleBot3