Building your own open source robot

Sid Faber

sid.faber@canonical.com

CANONICAL ubuntu

An Introduction to ROS

Building your own Open Source robot

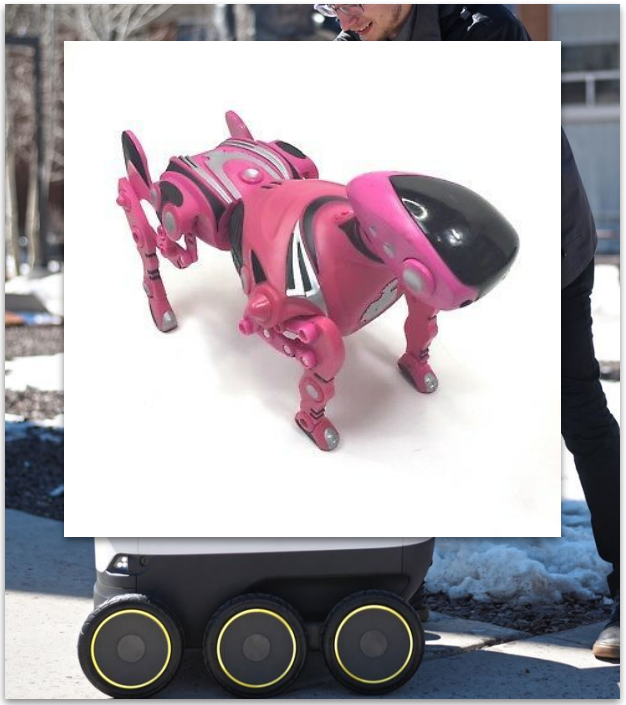
Part 1: ROS Basics

Sid Faber

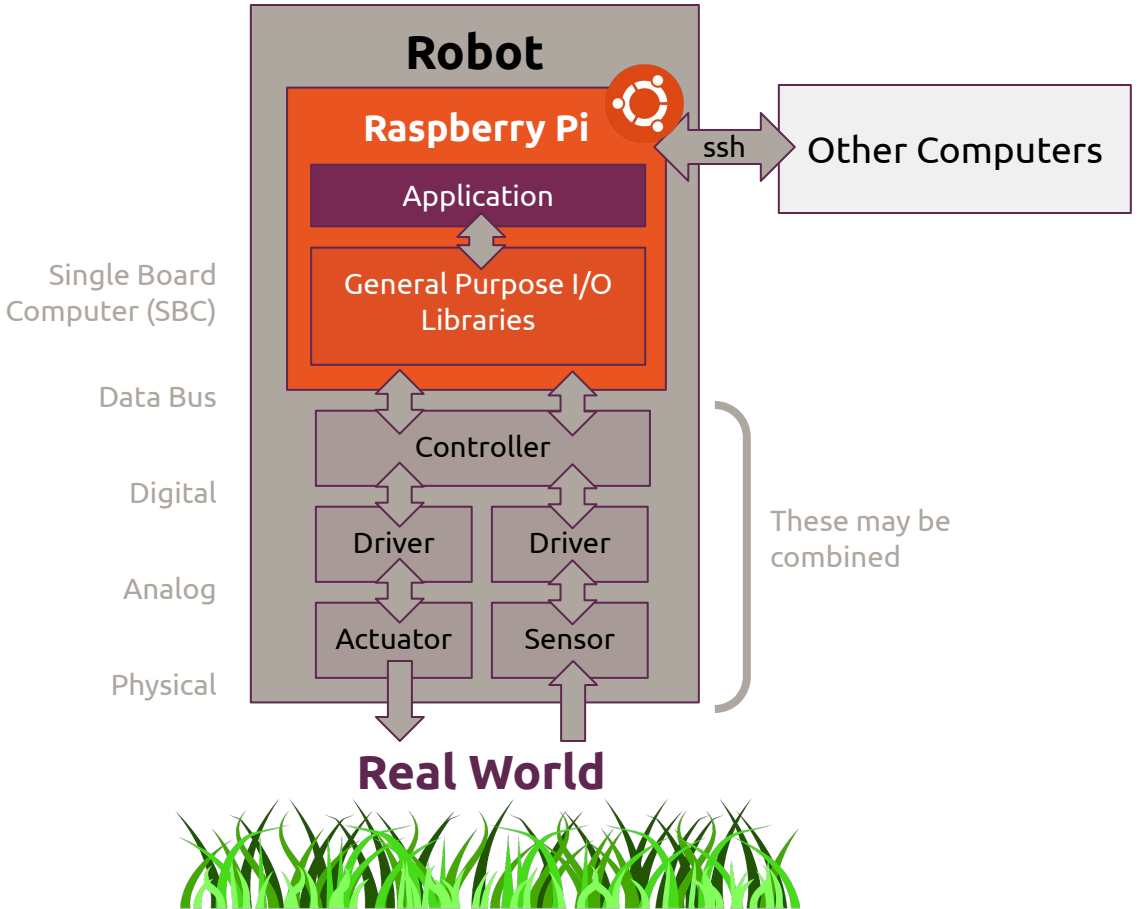
sid.faber@canonical.com

CANONICAL ubuntu

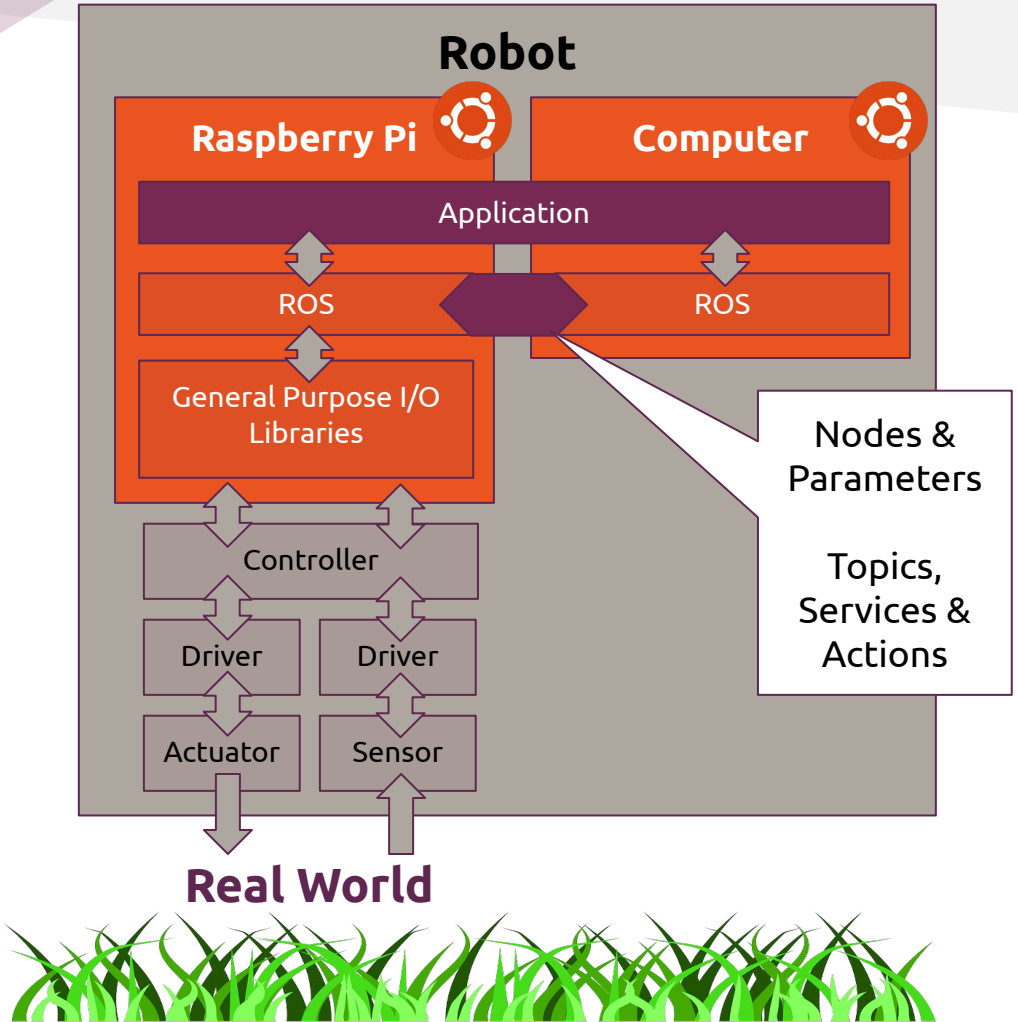
The current state of robotics



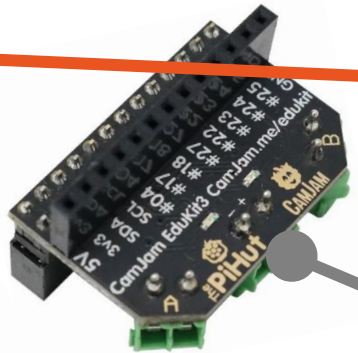
What is a [simple] robot?

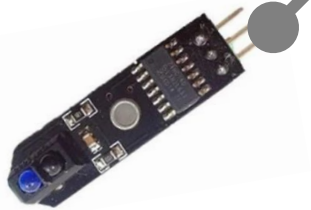


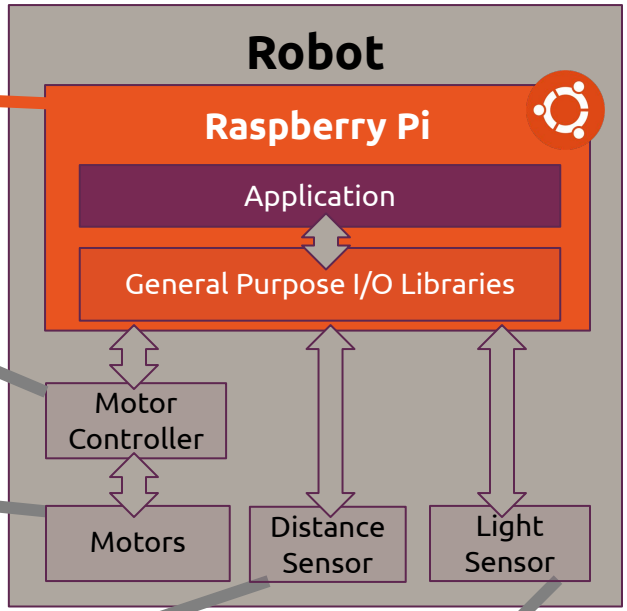
A ROS robot


Robot
Raspberry Pi
Computer
Application
ROS
ROS
General Purpose I/O Libraries
Nodes & Parameters
Topics, Services & Actions
Controller
Driver
Driver
Actuator
Sensor
Real World

Wheelie hardware: Motors and Sensors





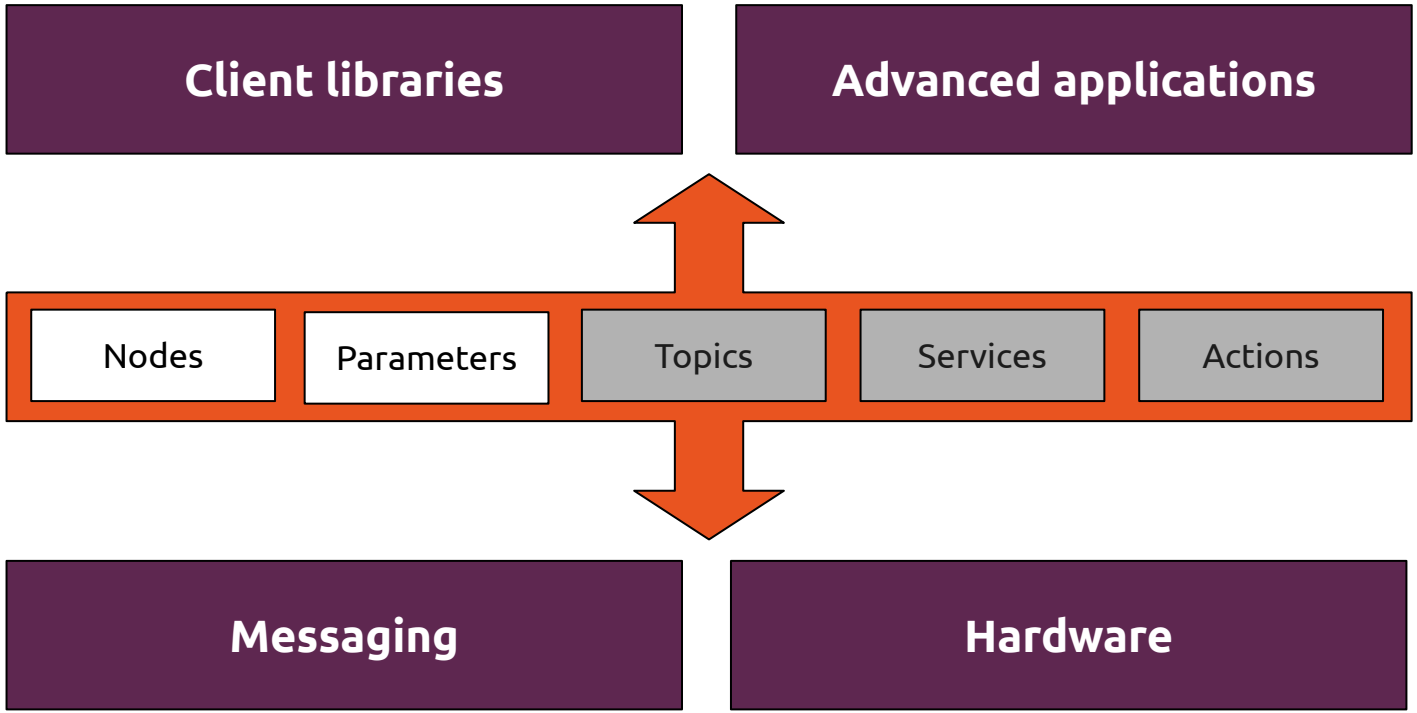


ROS architecture

ROS Structure

Nodes	A low level computational process
Parameters	Node runtime shared multi-variate dictionary
Topics	A named bus (pub/sub) over which nodes broadcast and receive messages (one-way)
Services	Request/response synchronous interaction with a node
Actions	Asynchronous goal request of a server, with periodic updates back to the client

ROS 2 middleware



An Introduction to ROS

Building your own Open Source robot

Part 2: Install the OS and put the parts together

Sid Faber

sid.faber@canonical.com

CANONICAL ubuntu

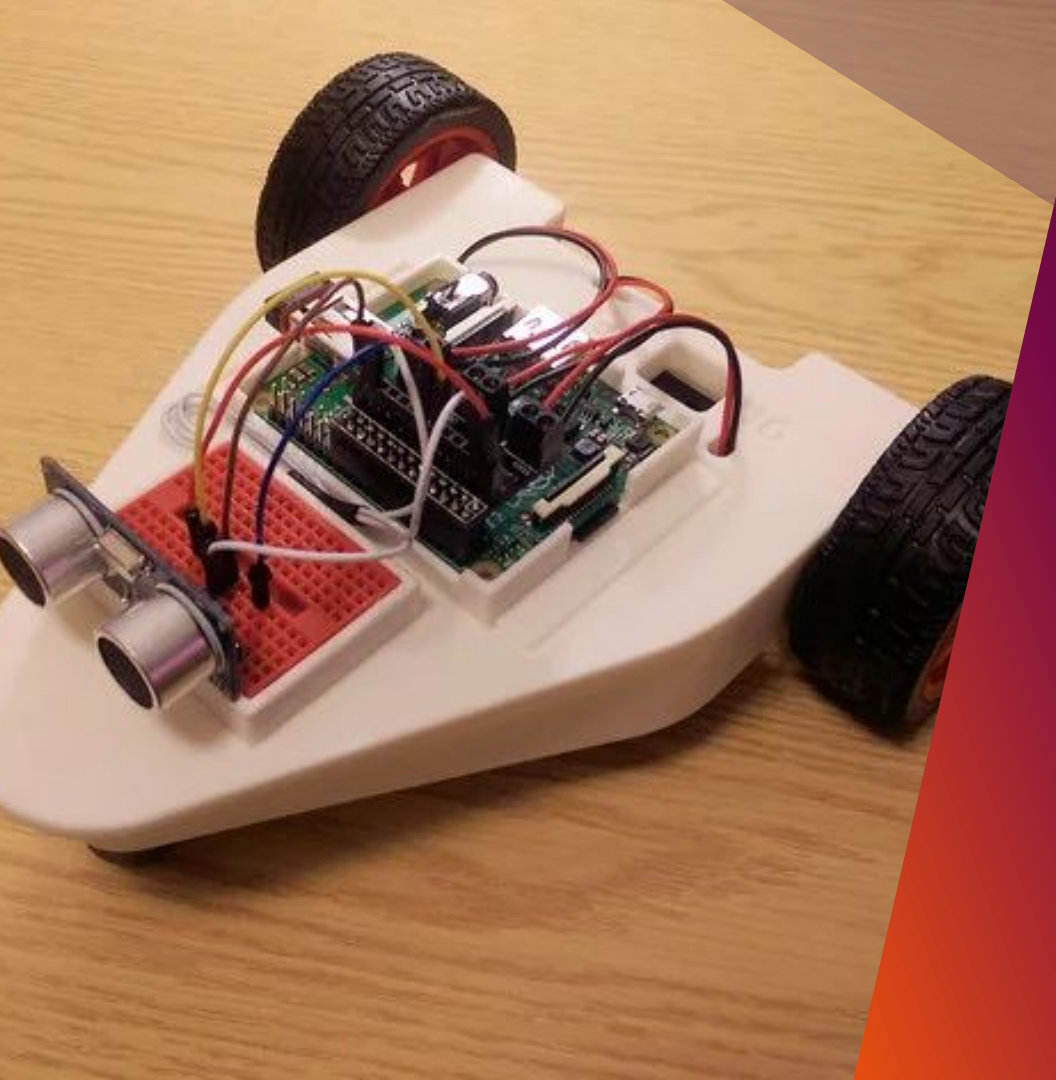
RasPi Build Steps

Install Ubuntu

Install ROS

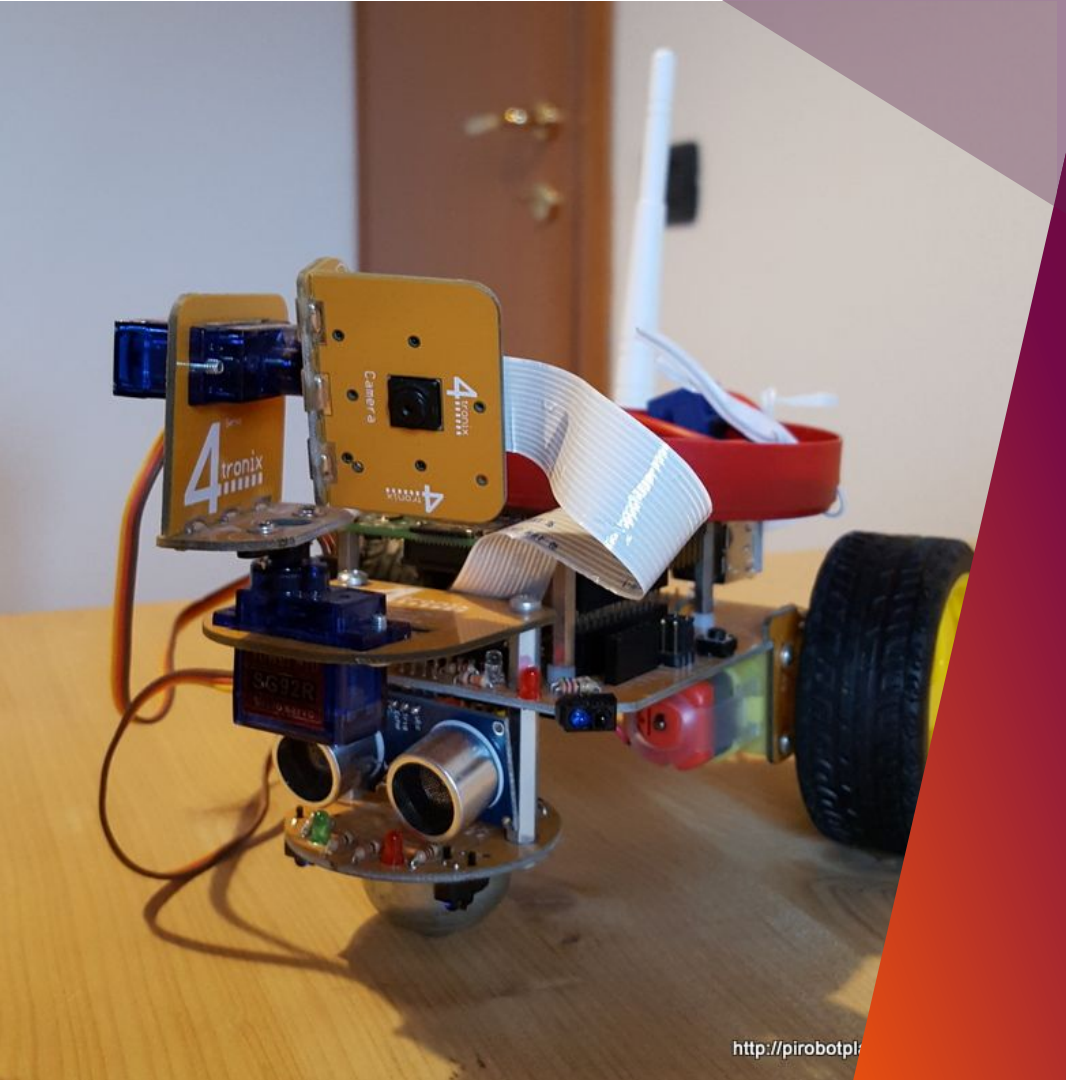
Clone the code repository

The Build



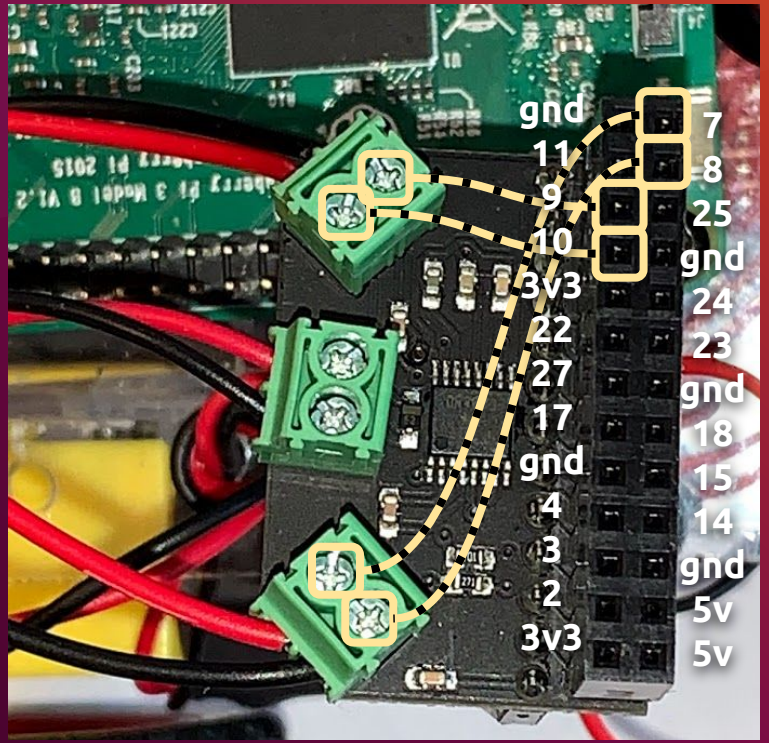
https://www.thingiverse.com/thing:1113796

https://github.com/fustinoni-net/PiRobotPlatform



https://github.com/fustinoni-net/PiRobotPlatform

Pin 7	Left Fwd
Pin 8	Left Back
Pin 9	Right Back
Pin 10	Right Fwd



An Introduction to ROS

Building your own Open Source robot

Part 3: Controlling hardware with ROS

Sid Faber

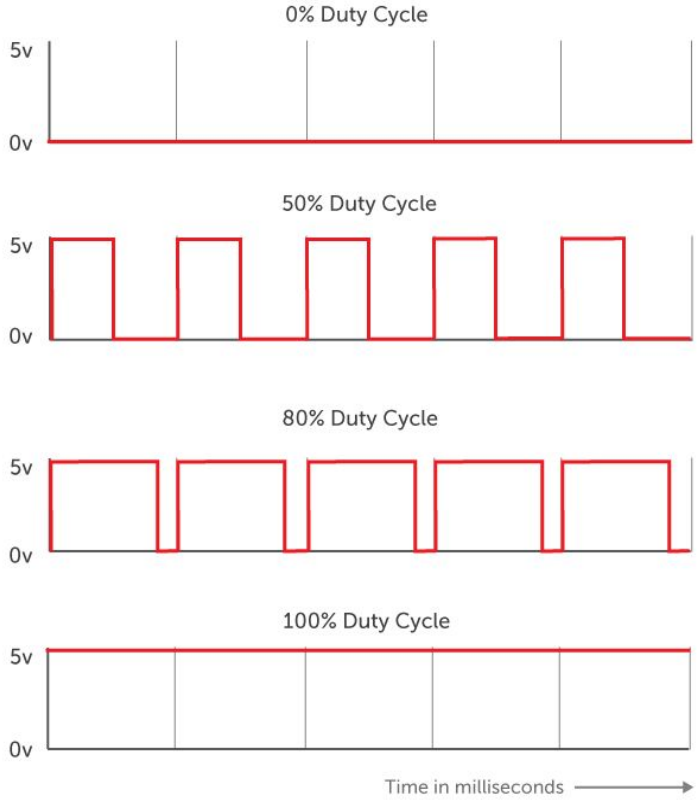
sid.faber@canonical.com

CANONICAL ubuntu

Move forward for one second

```
#! /usr/bin/python3
import RPi.GPIO as GPIO
import time

#  GPIO initialization
GPIO.setmode (GPIO.BCM)
GPIO.setwarnings (False)

#  set GPIO pin mode
GPIO.setup (7, GPIO.OUT)
GPIO.setup (8, GPIO.OUT)
GPIO.setup (9, GPIO.OUT)
GPIO.setup (10, GPIO.OUT)

#  turn motors off
GPIO.output(7,0)
GPIO.output(8,0)
GPIO.output(9,0)
GPIO.output(10,0)

#  turn right motor forward
GPIO.output(7,0)
GPIO.output(8,1)

#  turn left motor forward
GPIO.output(9,0)
GPIO.output(10,1)

time.sleep(1)

#  Reset GPIO and turn off motors
GPIO.cleanup()
```

Controlling motor speed: frequency and duty cycle




```
GPIO.setup (pinRFwd,  GPIO.OUT)
Frequency=20

#  set up PWM on pinRFwd at frequency 20
pwmRFwd  = GPIO.PWM (pinRFwd,  Frequency)

#  initialize pwm with a duty cycle of 0
pwmRFwd.start (0)

#  Change duty cycle to 0 after start
pwmRFwd.ChangeDutyCycle(0)

#  Change duty cycle to 50%
pwmRFwd.ChangeDutyCycle(50)

#  Change duty cycle to 80%
pwmRFwd.ChangeDutyCycle(80)

#  Change duty cycle to 100%
pwmRFwd.ChangeDutyCycle(100)
```

Image source: https://www.pubnub.com/blog/building-raspberry-pi-smart-house-part-2. For more about RPi.GPIO, see https://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/.

Adding ROS to the Motor class

```
#! /usr/bin/python3

import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import RPi.GPIO as GPIO
import time

GPIO.setmode (GPIO.BCM)
GPIO.setwarnings (False)

[include the Motor and Wheelie classes]

class MinimalSubscriber(Node):
    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'move',
            self.listener_callback,
            10)
        self.subscription  arning
        self.wheelie = Wheelie()
```

```
def listener_callback(self, msg):
    command = msg.data
    if command == 'forward':
        print('Moving forward')
        self.wheelie.goForward()
    elif command == 'backward':
        print('Moving backward')
        self.wheelie.goBackward()
    elif command == 'left':
        print('Turning left')
        self.wheelie.goBackward()
    elif command == 'right':
        print('Turning right')
        self.wheelie.goRight()
    elif command == 'stop':
        print('Stopping')
        self.wheelie.stop()
    else:
        print('Unknown command, stopping instead')
        self.wheelie.stop()
```

ros_helloworld.py

Adding ROS to the Motor class (2)

```
def main(args=None):
    #  initialize the wheelie node
    rclpy.init(args=args)
    minimal_subscriber = MinimalSubscriber()

    #  wait for incoming commands
    rclpy.spin(minimal_subscriber)

    #  Interrupt detected, shut down
    minimal_subscriber.wheelie.stop()
    GPIO.cleanup()
    minimal_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

ros_helloworld.py

Drive the Wheelie

On the Wheelie:

```
source /opt/ros/eloquent/setup.bash
./ros_helloworld.py
```

On your Laptop:

```
source /opt/ros/eloquent/setup.bash
ros2 topic pub '/move' \
     'std_msgs/String' \
     '{data: forward}'
```

...or run rqt to publish messages to the /move topic

An Introduction to ROS

Building your own Open Source robot

Part 4: Adding a joystick

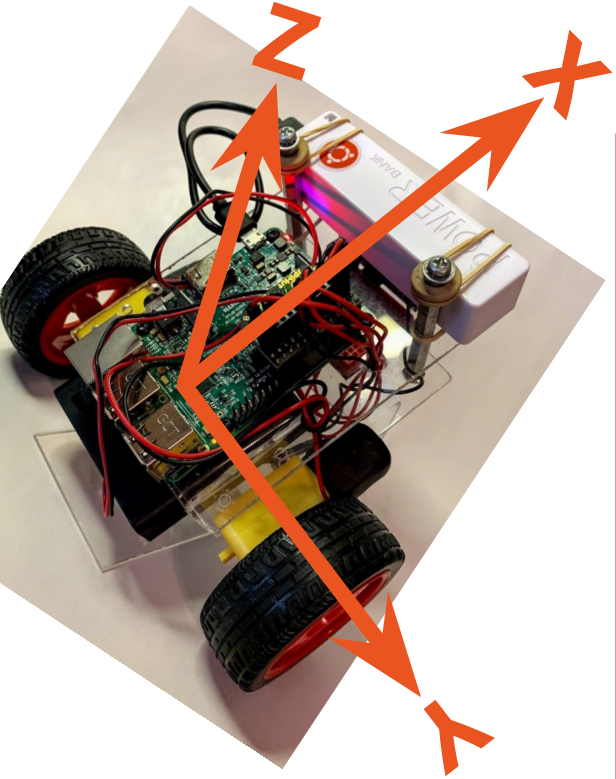
Sid Faber

sid.faber@canonical.com

CANONICAL ubuntu

Keyboard Control:

ros2 run teleop_twist_keyboard teleop_twist_keyboard



geometry_msgs/Twist

linear: Vector3

- x: forward meters/second
- y: left
- z: up

angular: Vector3

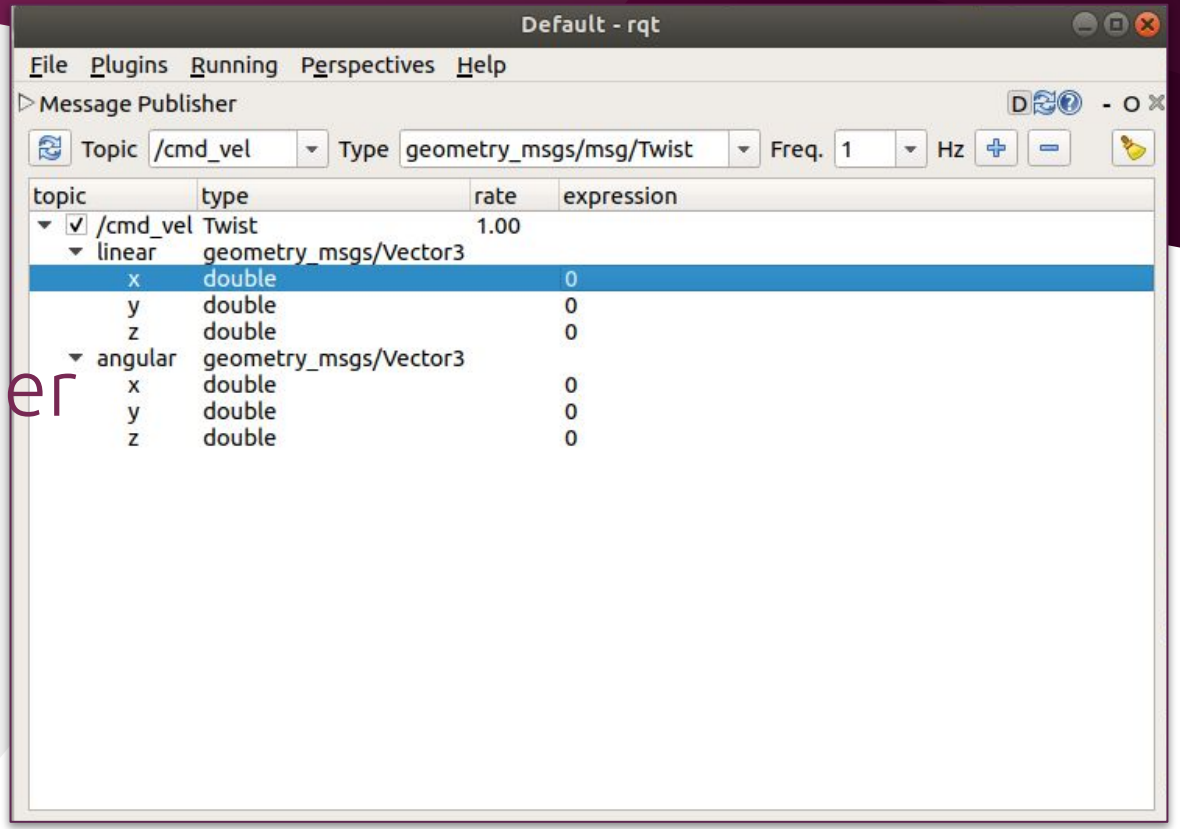
- x: radians/second
- y: radians/second
- z: rotation radians/second

Make it easy

ROS tools that work with Twist

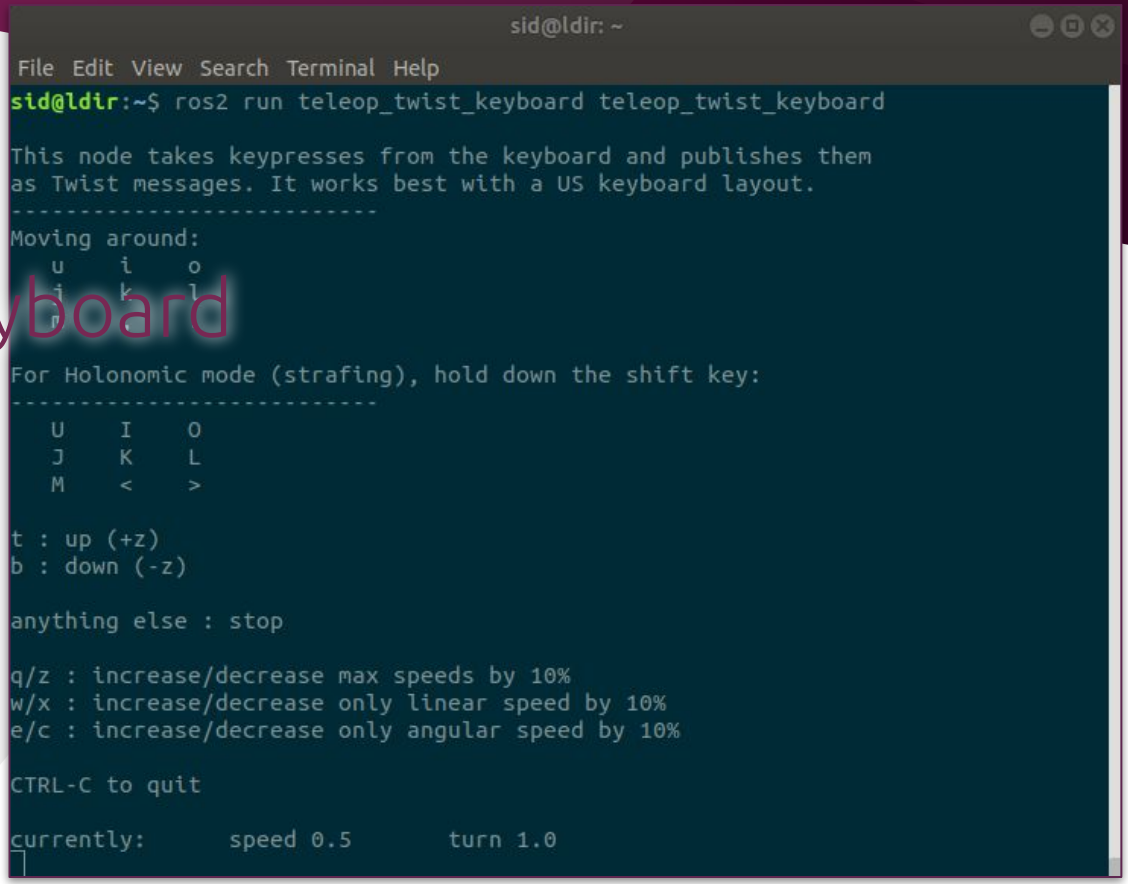
Moving the wheelie

rqt topic, Message Publisher plugin



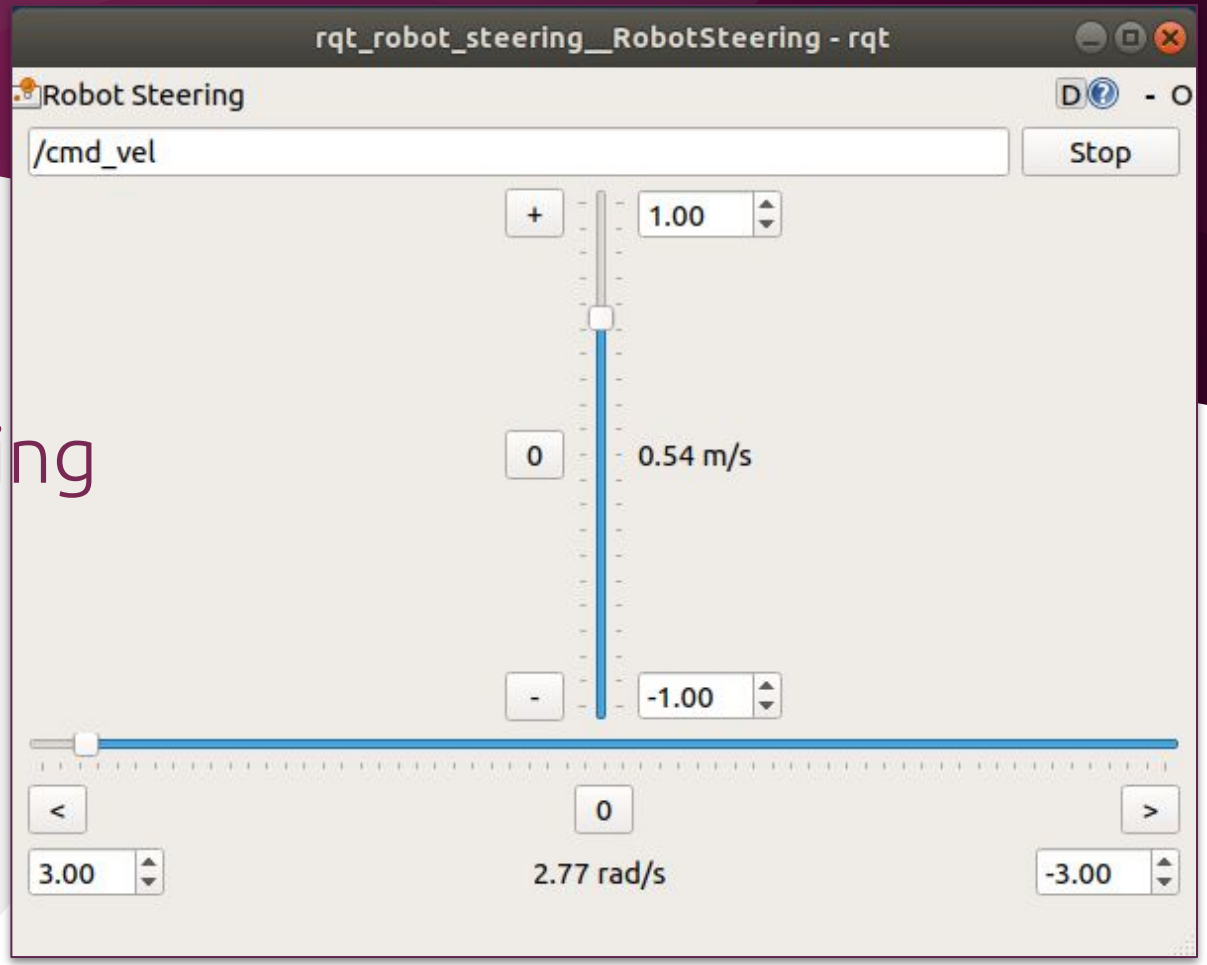
Moving the wheelie

teleop_twist_keyboard



Moving the wheelie

rqt_robot_steering



Joystick Control:

ros2 run joy joy_node

ros2 topic echo /joy

An Introduction to ROS

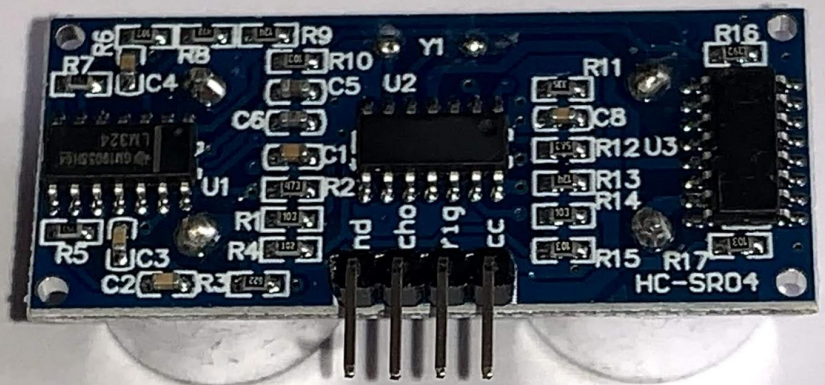
Building your own Open Source robot

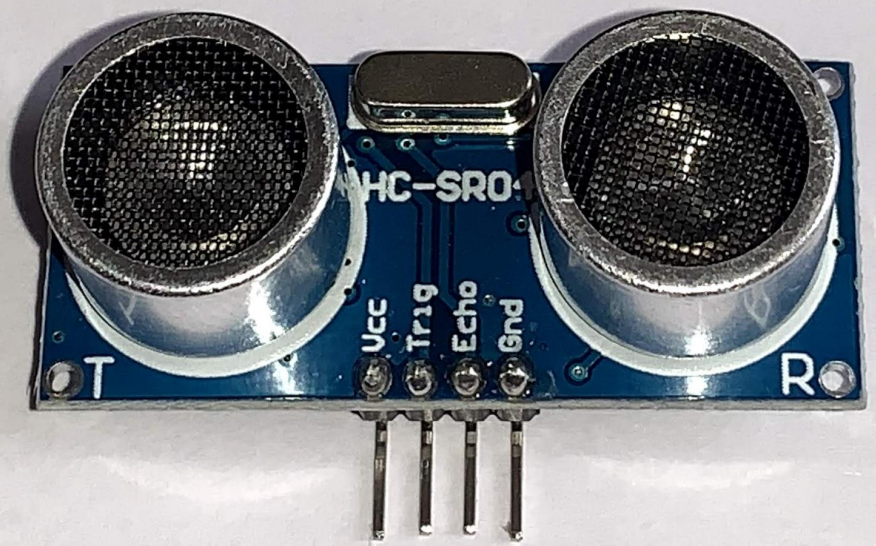
Part 5: Adding a sensor

Sid Faber

sid.faber@canonical.com

CANONICAL ubuntu


R9
R10
Y1
R16
R7
R8
C4
C5
U2
R11
C6
C8
C1
R12
U3
U1
R2
R13
R14
R1
R5
C3
R4
R15
R17
C2
R3
HC-SR04


HC-SR04
Vcc
Trig
Echo
Gnd
T
R

Building your own open source robot

Sid Faber

sid.faber@canonical.com

CANONICAL ubuntu